Learning Manifolds with the Parametrized Self-Organizing Map and Unsupervised Kernel Regression

Der Technischen Fakultät der Universität Bielefeld

vorgelegt von

Stefan Klanke

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften

März 2007

Gedruckt auf alterungsbeständigem Papier °° ISO 9706

Acknowledgements

First I would like to thank Helge Ritter for the supervision of my work and for making me start a PhD in the first place. Equipped with both an unbelievable schedule and a sense for fruitful research, he provided me with enough freedom for exploring my ideas, and he gave guidance whenever necessary.

Peter Meinicke came up with Unsupervised Kernel Regression in its original form and supervised my diploma thesis, and therefore laid the grounds for my dissertation – he definitely earned a huge "thank you" as well.

Sethu Vijayakumar, Ralf Möller, and Sven Wachsmuth made up my PhD committee besides Helge Ritter. Triple cheers to them for agreeing to review this rather long and dry work!

My thanks also go to the whole Neuroinformatics group for forming a nice place to work, especially to Petra Udelhoven for keeping everything running smoothly, and to Jan Steffen for proof-reading large parts of my thesis. Katharina Tluk v. Toschanowitz and Stefan Krüger created a beautiful doctoral hat and provided support for my disputation, which I really appreciated.

I would also like to thank my fellow PhD students from the graduate programme "Strategies and Optimization of Behaviour", in particular Nadine Reinhold and Sina Kühnel – our breakfast and coffee breaks often made my day.

Further thanks are in order for my parents, who supported me throughout my studies, and my friends, especially Ivi and Holger, for keeping me busy with things beyond machine learning.

Last but not least I would like to express my gratitude to my dear girlfriend Anika, who enriched my life and made sure I kept going in stressful times.

This work was funded by a DFG scholarship (GK 516).

Abstract

This thesis presents several new developments in the field of manifold learning and nonlinear dimensionality reduction. The main text can be divided into three parts, the first of which presents a smoothness-based regularizer that is specifically tuned to the Parametrized Self-Organizing Map (PSOM). The regularization approach makes it possible to deal with noisy or missing data in a principled manner, and it facilitates the construction of PSOMs from data that are not organized in a grid topology.

In the second part, the manifold learning algorithm Unsupervised Kernel Regression (UKR) is introduced as a counterpart to the classical Nadaraya-Watson estimator. In a nutshell, UKR requires very little parameters to be chosen a priori: In its simplest form, a UKR model is fully specified by the dimensionality of latent space and the choice of a density kernel, and it can be regularized automatically by using leave-one-out cross-validation without additional computational cost. The low dimensional coordinates (latent variables) together with a mapping from latent space to data space are retrieved by minimizing some error criterion.

The third part presents four possible extensions to UKR, specifically 1) a more general cross-validation scheme, aimed at avoiding unsmooth manifolds, 2) the inclusion of loss functions beyond the usual squared error, which can enhance the robustness towards outliers, and by which UKR can be tuned to specific noise levels, 3) a "landmark" variant which helps to reduce the computational cost, and 4) Unsupervised Local Polynomial Regression, where the Nadaraya-Watson estimator is replaced by local linear or local quadratic regression models, the latter showing less bias in the presence of curvature.

Contents

1. Introduction

1.1. Motivation

The last decades have brought up a massive increase of computing power and data storage facilities. Nowadays, even standard home computers are able to carry out billions of elementary operations per second. At the same time, the amount of data that is collected and processed electronically has also increased, typically with the desire to use that data for improving businesses or services, and of course also for research purposes e.g. in bioinformatics. Unfortunately, humans did not scale as well as computers, so there is a huge need for "intelligent" automated data processing, that is, the data has to be analyzed and processed with as little human guidance as possible, until it is structured or simplified to a level that is accessible at least to human experts.

In some fields of application, human intervention is not possible or reasonable at all, and so the data processing system has to work fully autonomously. As an example one may think of robots exploring the Mars, or of more earthly situations where robots carry out work that is too dangerous for humans, for example removing land mines.

To address the aforementioned problems, the field of *machine learning* is concerned with providing algorithms that allow computers to "learn" in the sense of generating useful models or representations from a series of observations. Machine learning algorithms can be broadly differentiated by the nature of the observations they process, where for this thesis only the two classes of supervised and primarily unsupervised learning are relevant and will be described later.

In any case, even if a data processing system operates without human intervention, a human expert will have decided *how* the system learns by the very choice of its implementation. As a consequence, for a general applicability of the system, the learning algorithm should ideally not depend on certain adjustments of parameters that are hard or even impossible to automatically estimate from the observations. The central topic of this thesis is Unsupervised Kernel Regression, a recently introduced manifold learning algorithm that was specifically designed in that way.

1.2. Outline and contributions

The remainder of this chapter provides a brief introduction to supervised and unsupervised learning, primarily manifold learning, as well as an explanation of further concepts and techniques that will be required later on. An overview of related work on manifold learning is then given in chapter 2.

Chapter 3 is devoted to the description of the Parametrized Self-Organizing Map and several extensions to the algorithm contributed by the author of this thesis. Parts of this chapter have already been published (Klanke and Ritter, 2005).

In chapter 4, the manifold learning algorithm Unsupervised Kernel Regression (UKR) will be presented. The algorithm in its original form has been conceived mainly by Peter Meinicke, but first of all large parts of the calculations, the implementation, and the experimental evaluation have been contributed by the author. Parts of chapter 4 have already been published (Meinicke, Klanke, Memisevic, and Ritter, 2005).

Chapter 5 contains the description and experimental evaluation of several extensions to the original UKR algorithm, also contributed by the author. The first two extensions correspond to already published work (Klanke and Ritter, 2006b, 2007, 2006a). The thesis will be concluded with a discussion of the contributions in chapter 6.

In the following, for easier readability the pronoun "we" will be utilized throughout the text, whereas it can stand for a textbook-like inclusion of the reader or indicate work that was carried out by the author alone or in collaboration with others.

1.3. Supervised learning

Supervised learning is the generic term for modelling the relationship between input data (also called *predictors*, or *independent variables*) and corresponding output data (also called *responses*, or *dependent variables*). Friedman (1994, p. 7) points out "two distinct practical reasons" for applying supervised learning techniques. The first is *prediction*, where one wishes to utilize the learned relationship to properly respond to new input data. An example of this kind of usage is a spam filter, which automatically decides if an incoming email should be treated as junk or be actually presented to the recipient.

The other reason is *interpretation*, where the properties of the relationship itself are of interest. For example, when the data stems from observing a physical system, the obtained knowledge about the relationship may help to better understand that system.

Depending on the nature of the output data, one also differentiates between the tasks of

- *regression*, where the output data are quantitative (usually real-valued) measurements, and
- *classification*, where the output data are qualitative or categorical variables, e.g., a class label like "apple","pie" and "orange". Quite often either the output data is already binary (e.g., "valid" vs. "invalid"), or the multi-class problem is tackled by a combination of binary classifiers (e.g., Allwein, Schapire, and Singer, 2000).

As stated by Hastie, Tibshirani, and Friedman (2001, p. 10), "these two tasks have a lot in common, and in particular both can be viewed as a task in function approximation." Nevertheless, since classification does not play an important role in this thesis, the remainder of this section focuses on regression and its statistical foundations.

1.3.1. Statistical model

In the literature on regression (e.g., Bates and Watts, 1988), the input and output data are treated as realizations of random variables, which takes into account that any system and any kind of measurement is usually subject to small fluctuations or pertubations.

Let us assume we have two real-valued random variables X (input) and Y (output) with a given joint distribution $p(x, y) = p(X = x, Y = y)$. We seek a function $f(\cdot)$ which optimally predicts the output y for a given input x. For this, we first have to specify a *loss function* $L(y, f(x))$ which penalizes errors in our prediction. The most common choice for this is the squared error loss

$$L(y, f(x)) = (y - f(x))^2 . \tag{1.1}$$

We can now judge the quality of a mapping $f(\cdot)$ by calculating the *expected prediction error* functional

$$E_{exp}(f) = \langle L(Y, f(X)) \rangle = \iint L(y, f(x))\, p(x, y)\, dx\, dy. \tag{1.2}$$

If we factor the joint density using the conditional density of Y given X, i.e.

$$p(x, y) = p(y|x)\, p(x) = p(Y = y|X = x)\, p(X = x) \tag{1.3}$$

and insert the squared error loss, we get

$$E_{exp}(f) = \int \left(\int (y - f(x))^2\, p(y|x)\, dy \right) p(x)\, dx. \tag{1.4}$$

Minimizing E_{exp} with respect to $f(\cdot)$ can thus be carried out by pointwise minimization of the inner integral for every x, which in turn yields that the optimal mapping $f^*(\cdot)$ equals the conditional expectation

$$f^*(x) = \int y\, p(y|x)\, dy = \langle y|x \rangle. \tag{1.5}$$

This expression is also called *regression function*. Its form suggests to view the output data as a superposition of a systematic part, which depends on the input data, and of additive noise, which has zero mean and is independent of the input data. This can be written as the *regression model*

$$y = f^*(x) + u \quad , \quad \langle u \rangle = 0. \tag{1.6}$$

All equations can easily be generalized to the case of multivariate input and output variables $\mathbf{x} \in \mathbb{R}^q$ and $\mathbf{y} \in \mathbb{R}^d$, where the squared error loss (1.1) becomes the squared Euclidean distance (please see appendix A for an overview of the notation)

$$L(\mathbf{y}, \mathbf{f}(\mathbf{x})) = \|\mathbf{y} - \mathbf{f}(\mathbf{x})\|^2. \tag{1.7}$$

1.3.2. Empirical error minimization and regularization

As demonstrated in the last section, the optimal regression function can theoretically be computed from the joint distribution $p(\mathbf{x}, \mathbf{y})$ of the input and output variables. In a typical learning task, however, this distribution is not known, but rather only a finite number of independent samples from it, which together make up the set of training data

$$\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) \;\; | \;\; i = 1 \ldots N\}. \tag{1.8}$$

Still, the task is to model the relationship between the input and output data, that is, we seek a function $\mathbf{f}(\cdot)$ which maps a given $\mathbf{x}$ to the correct $\mathbf{y}$. In order to measure the success of $\mathbf{f}(\cdot)$, we again use a loss function to quantify the prediction error, but this time we are restricted to the finite set $\mathcal{D}$ of examples. Thus, for the case of the squared error loss, we would calculate

$$E_{emp}(\mathbf{f}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i)\|^2, \tag{1.9}$$

which is known as the *empirical error* or *empirical risk*. Unfortunately, simply choosing that function $\mathbf{f}(\cdot)$ which minimizes (1.9) without further restrictions is not reasonable. Since there is an infinite number of continuous functions which can interpolate the training data with an otherwise arbitrary behavior, the learning problem is ill-posed (e.g., Cherkassky and Mulier, 1998). Furthermore, looking back at the two practical reasons for supervised learning (Sec. 1.3), it is clear that $\mathbf{f}(\cdot)$ should not only give the correct mapping for the input data $\{\mathbf{x}_i\}$ it is estimated from. Rather, it should have the ability of *generalization*, that is, it should properly predict the output for new input data (drawn from the same distribution) and/or it should contribute to the understanding of a system under study.

Statistical learning theory, the foundations of which were developed in the 1960s by Vapnik and Chervonenkis, offers a wealth of concepts and results regarding the conditions under which empirical risk minimization works. A detailed description of these concepts is beyond the scope of this section, and therefore is left to the textbooks of Vapnik (1998, 1995).

Two important concepts shall be mentioned nonetheless. The first concept is *consistency*, where one analyses the behavior of an empirical risk minimization algorithm in the limit of an infinite number of training examples. An algorithm is said to be consistent, if in this limit the empirical error converges uniformly against the expected error (cf. Fig. 1.1).

The second (related) concept is about controlling the *capacity* of a learning system or a function model. Here, one restricts the class $\mathcal{F}$ of functions from which $\mathbf{f}(\cdot)$ is selected. As a simple example, consider the class of polynomials up to a specific degree. If the degree is chosen smaller than the number of training examples, arbitrary interpolation as mentioned above is not possible anymore. Thus, the learning system is forced to use its remaining flexibility as well as possible, which hopefully results in a better generalization performance, that is, a lower expected error as depicted in Fig. 1.2.

Besides directly restricting the function class, one can add a *penalty term* $P(\mathbf{f})$ to the empirical risk (1.9), which has small values for smooth functions and larger values for increasingly complex functions. This results in the *regularized risk*

$$E_{reg}(\mathbf{f}, \lambda) = E_{emp}(\mathbf{f}) + \lambda P(\mathbf{f}) \tag{1.10}$$

and – together with a restricted function class – in the minimization problem

$$\hat{\mathbf{f}} = \arg\min_{\mathbf{f} \in \mathcal{F}} E_{reg}(\mathbf{f}, \lambda) \ , \tag{1.11}$$

where the parameter λ balances between the rivalling goals of a smoother function and a smaller error on the training data.

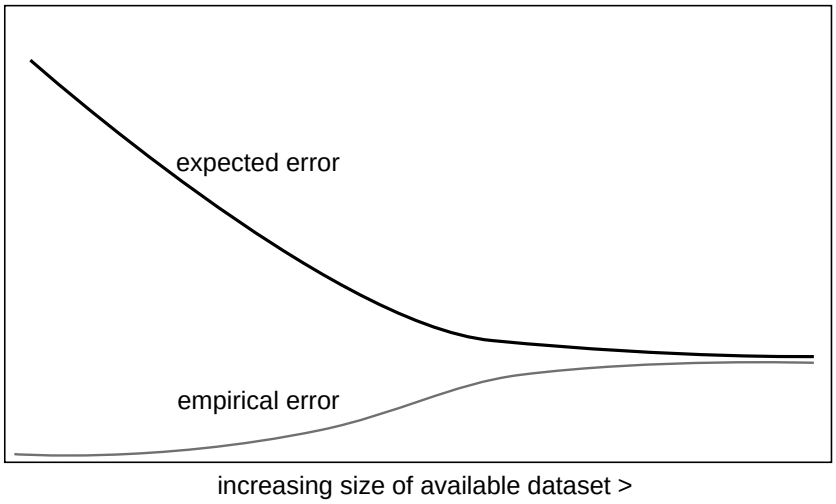


Figure 1.1: Schematic illustration of the relation between empirical and expected error as a function of the size of the available dataset.

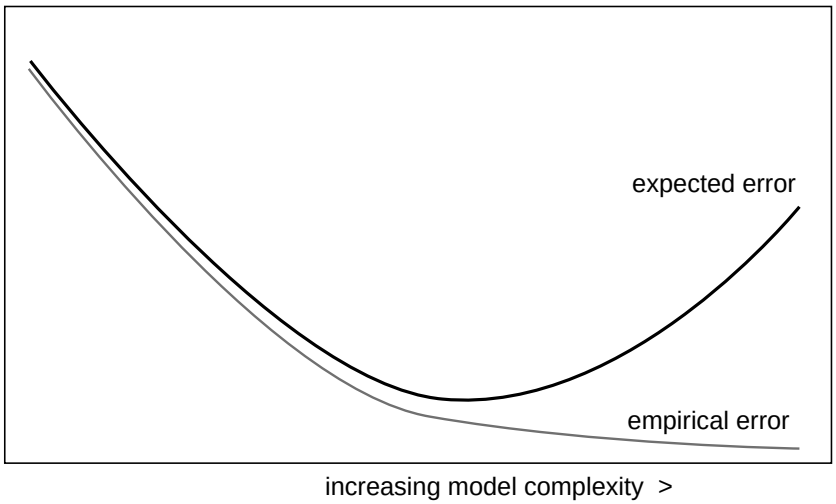


Figure 1.2: Schematic illustration of the relation between empirical and expected error as a function of the model complexity.

1.3.3. Bias variance dilemma

A related important concept in statistical learning is the so-called *bias variance dilemma*, which for example Bishop (1995) describes in detail. Briefly summarized, one analyses how much a function[1] $\hat{f}$, which is estimated on a finite dataset, typically differs from the true regression function f^* (that is, the conditional expectation). In particular, the expected squared error for predicting $f^*(\mathbf{x})$ can be decomposed and written as

$$\left\langle (\hat{f}(\mathbf{x}) - f^*(\mathbf{x}))^2 \right\rangle_{\mathcal{D}} = \underbrace{\left(\langle \hat{f}(\mathbf{x}) \rangle_{\mathcal{D}} - f^*(\mathbf{x}) \right)^2}_{(\text{bias})^2} + \underbrace{\left\langle (\hat{f}(\mathbf{x}) - \langle \hat{f}(\mathbf{x}) \rangle_{\mathcal{D}})^2 \right\rangle_{\mathcal{D}}}_{\text{variance}}, \tag{1.12}$$

[1]For simplicity, the description is restricted to the case of univariate output data.

where $\langle\cdot\rangle_{\mathcal{D}}$ denotes an average across all datasets. The bias describes the deviation between the "average" estimated $\hat{f}(\cdot)$ and the true regression function. This deviation does not depend on a particular training set $\mathcal{D}$, but rather a large bias indicates an overly simplistic function model. The variance, however, measures the spread of the estimated functions $\hat{f}(\cdot)$, which in turn measures how much the estimation of a single $\hat{f}(\cdot)$ depends on the data at hand.

The dilemma now lies in the fact that, if the available data is limited, one cannot reduce the bias (e.g., by allowing complex functions) without increasing the variance, and vice versa. This is illustrated in Fig. 1.3.

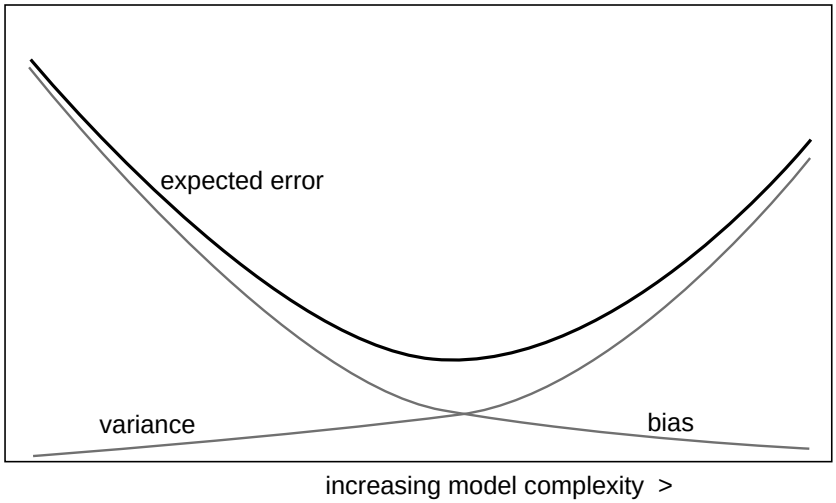


Figure 1.3: Schematic illustration of the bias variance dilemma.

1.3.4. Model selection

We now turn our attention from theoretical considerations regarding bias, variance and the behavior in the limit of infinite data to the practical problem of selecting and fitting a model in a concrete learning task.

Given a particular dataset $\mathcal{D}$, how can one reach a good trade-off between *overfitting*, which denotes the case of estimating a too complex function (corresponding to the small bias, large variance case), and *underfitting*, which denotes the opposite? In other words, how do we select parameters of the function class (e.g., the degree of polynomials) and regularization parameters, for example λ in (1.10) ?

A simple practical approach is given by the *hold-out method*. Instead of using the complete dataset $\mathcal{D}$ to train the learning system, that is, to estimate the desired function $\hat{f}(\cdot)$, a subset $\mathcal{D}_{val}$ is held out and the system is trained with the remaining data $\mathcal{D}_{train} = \mathcal{D}\backslash\mathcal{D}_{val}$. Different regularization settings can then be compared by the system's prediction performance on the *validation* set $\mathcal{D}_{val}$, which serves as an estimation of the expected prediction error (1.2).

The main problem of this approach is that the number of training examples is reduced, which theoretically worsens the estimate. This is particularly relevant in applications

where the available dataset is already small from the start. Furthermore, instead of overfitting the training set, the learning system might now be tuned too much to the validation set. Thus, for a final judgement of the trained learning system, the expected prediction error has to be estimated from a third subset of the data, the "test set". Typically, the partitioning of the data is chosen so that the training set contains 50% percent, the validation set contains 25%, and the test set contains the remaining 25% of the data (Hastie et al., 2001).

A better, but usually costlier method is given by *cross-validation* (M. Stone, 1974, 1977), which also works by partitioning the given data $\mathcal{D}$, albeit using a more complex scheme as compared to the hold-out method. In particular, *K-fold* cross-validation consists of splitting the data into K disjoint subsets $\mathcal{D}_1, \mathcal{D}_2 \ldots \mathcal{D}_K$ of roughly equal size. For each adjustment of the regularization parameters, the learning system is trained K times on the K datasets $\mathcal{D} \backslash \mathcal{D}_i,\ i = 1 \ldots K$ and its prediction performance is evaluated on the left-out set $\mathcal{D}_i$. The best average performance then determines the choice of the parameters.

Through this, the complete dataset takes part in both training and validation of the system. Higher values of K usually lead to better estimates, at the price of an increased computational effort. The limit of $K = |\mathcal{D}|$, that is, N-fold cross-validation for a dataset consisting of N examples, is also called leave-one-out cross-validation (LOO-CV). While generally being the most "expensive" variant, in some cases LOO-CV can actually be applied at the cost of a single learning run. Unsupervised Kernel Regression, as will be introduced in chapter 4, is among these cases.

As an alternative to cross-validation, model selection can also be achieved by help of the *bootstrap* method (Efron, 1979). Bootstrapping works by generating new datasets by sampling with replacement from the original dataset. Thus, the new datasets might not include some original elements, while others are included multiple times. In some studies (Efron, 1983; Lendasse, Wertz, and Verleysen, 2003), bootstrapping is reported to be superior to cross-validation, other studies (Kohavi, 1995) deny this. A fast bootstrap method, which requires comparatively few learning runs, has been proposed recently (Lendasse, Simon, Wertz, and Verleysen, 2005).

Besides these general approaches to model selection, further methods are applicable when using special learning strategies. In particular, the Akaike information criterion and Bayesian inference can be utilized in maximum likelihood and Bayesian estimation, respectively (Hastie et al., 2001; Bishop, 1995). These learning strategies will be briefly discussed in section 1.5.3.

1.4. Unsupervised learning

The supervised learning problem, as discussed in the last section, can be characterized as a function approximation task. Through the distinction between input and output data, both the goal and the success of learning can be stated clearly. The given output data $\{\mathbf{y}_i\}$ can be viewed as a teacher signal that tells the learning system which response it should give to any presented example of input data $\{\mathbf{x}_i\}$.

As the name implies, in unsupervised learning, there is no teacher signal, and also the learning goal and its measure of success are far less clear. Hastie et al. (2001, p. 2)

state: "In the *unsupervised learning problem*, we observe only the features and have no measurements of the outcome. Our task is rather to describe how the data are organized or clustered."

Hertz, Krogh, and Palmer (1991) point out that the existence of redundancy in the data is a necessary condition for the application of unsupervised learning techniques. As a possible learning goal, they list *clustering* (see, e.g., Jain and Dubes, 1988 or Hartigan, 1975), where each data sample is assigned a discrete value – similar to a class label – or a *prototype*, which refers to a typical representative of such a "class" (see Sec.. 2.1). Further listed goals are the extraction of *principal components* from the data (see Sec. 2.2), finding an *encoding* of the data with fewer bits (e.g., Gersho and Gray, 1992) and finally the creation of a topographic feature map (see Sec. 2.4).

Ripley (2005, p. 2) describes the task of unsupervised learning as the "discovery of new groupings" and later gives an overview of unsupervised methods, where he focusses on the visualization of high-dimensional data.

In fact, the data being high-dimensional is very typical of unsupervised learning tasks, and all aforementioned learning goals can be subsumed as *finding an alternative, more compact representation of the data*. The need for finding such a representation is a direct consequence of the unfavorable properties of high-dimensional spaces, which Bellman (1961) called the "curse of dimensionality". Imagine you wish to scan a d-dimensional hyper-cube with 10 steps per dimension. For $d = 2$ (a square), you need 100 samples, for $d = 3$ (a normal cube), you already need 1000 samples. With any further dimension, the required number of samples rises exponentially. The other way around, imagine a dataset consisting of 10,000 points in 6 dimensions. If you put the 10-step rastered 6-D hyper-cube around the data, most ($\approx 99\%$) volume elements will be empty.

1.4.1. Dimension reduction and manifold learning

In many applications, the dimensionality of the observed data (the length of the data vectors) does not reflect their *intrinsic dimensionality*, but is much higher. A series of pictures taken of a rotating object, for example, has an intrinsic dimensionality of $d_{int} = 3$, because the object has three rotational degrees of freedom. The resulting images, however, might have a resolution of 640×480 pixels, and thus correspond to $640 \cdot 480 = 307,200$-dimensional data vectors, where each entry encodes the intensity of one pixel. Because of the way they were generated, the data vectors do not populate the data space evenly, but lie on an *embedded manifold*, that is, they form a locally low-dimensional (in this case, 3-dim.) structure.

To illustrate the concept of an embedded manifold, a simple example is depicted in Fig. 1.4. The left plot shows 100 data points in two dimensions. The points are not evenly distributed across the whole plane, but rather lie along a curved path. In fact, the data were generated by sampling, with noise, from the curve which is depicted in the right plot of Fig. 1.4. This curve can be viewed as an embedded 1-D manifold in the 2-D space of the observed data.

In this case, the description of the data points by their 2-D coordinates contains redundancy. If we ignore small deviations of the data points from the underlying curve, the data is essentially one-dimensional. A more compact and, at the same time, more revealing representation of the dataset would be given by the positions of the data points

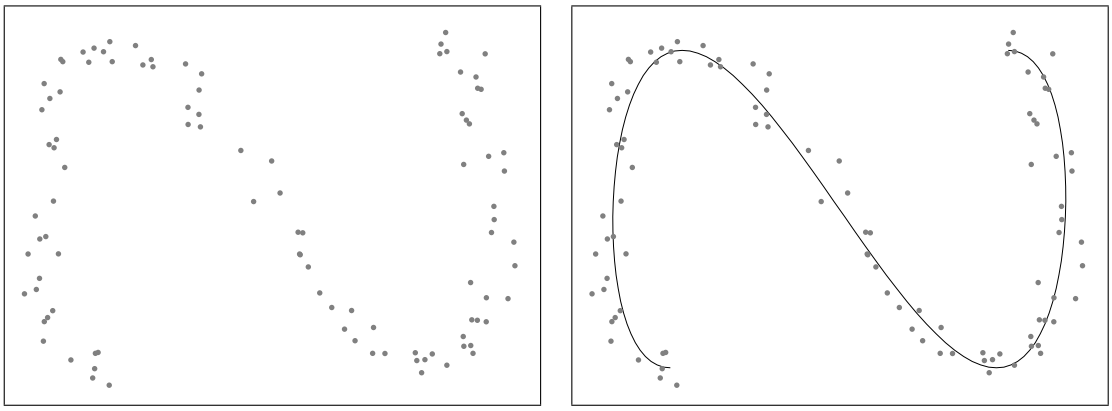


Figure 1.4: Example of a dataset in 2-D, which lies roughly on an embedded 1-D manifold. The data points were generated by sampling from the black curve and adding some noise.

along the curved path, or rather the positions of their projections onto the curve, as depicted in Fig. 1.5.

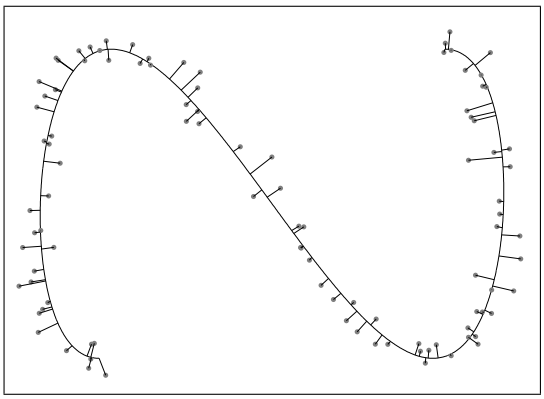


Figure 1.5: Projection of a 2-D dataset onto the underlying 1-D manifold.

Of course, normally the form of an underlying manifold is not known, but rather has to be estimated from the data. This has to be done in conjunction with finding the "hidden" lower dimensional representations of the data. As Meinicke (2000) points out, this task can be viewed as a generalized regression problem, where the observed data vectors represent the output data $\mathbf{y}_i$, while their unknown lower dimensional representations take the role of the input data $\mathbf{x}_i$. Similar to the regression model (1.6), one then views a data vector $\mathbf{y}_i \in \mathbb{R}^d$ as being "explained" by a superposition of 1) a systematic part, that is, a lower dimensional vector $\mathbf{x}_i \in \mathbb{R}^q$ and its mapping $\mathbf{f}(\mathbf{x}_i)$ into data space, and 2) zero-mean noise, which represents fluctuations and inaccuracies:

$$\mathbf{y}_i = \mathbf{f}(\mathbf{x}_i) + \mathbf{u} \ , \quad \langle \mathbf{u} \rangle = 0. \tag{1.13}$$

If one treats the vectors $\mathbf{y}_i$ and $\mathbf{x}_i$ as realizations of random variables, this approach coincides with *latent variable* modelling (Everitt, 1984). In the following, the lower dimensional representations $\mathbf{x}_i$ will also be called "latent variables", where the difference between a random variable and its realizations is ignored.

A possible criterion for finding the "right" latent variables $\{\mathbf{x}_i\}$ and the corresponding mapping $\mathbf{f}(\cdot)$ is given by the empirical reconstruction error

$$R(\{\mathbf{x}_i\}, \mathbf{f}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i)\|^2. \tag{1.14}$$

Similarly to the supervised regression case, a direct minimization of this error measure without restrictions on the function class or the latent variables is not reasonable, and some means of regularization is needed. Unsupervised Kernel Regression, as will be demonstrated in chapter 4, features a particularly elegant approach to this problem.

Several manifold learning algorithms only aim at retrieving the lower dimensional representations $\mathbf{x}_i$ without supplying an explicit model of the manifold as given by the mapping $\mathbf{f}(\cdot)$. Such algorithms are typically oriented towards geometric properties, and for example demand that the distances $d(\mathbf{y}_i, \mathbf{y}_j)$ between two data vectors $\mathbf{y}_i$ and $\mathbf{y}_j$ should be as close as possible to the distances $d(\mathbf{x}_i, \mathbf{x}_j)$ between their corresponding latent space representations $\mathbf{x}_i$ and $\mathbf{x}_j$.

The most important algorithms for manifold learning and dimension reduction are reviewed in chapter 2.

1.5. Further concepts

This section introduces some further basics that will be needed in later parts of this thesis. The concepts herein apply to both supervised and unsupervised learning.

1.5.1. Density estimation

A particularly good description of data is given by their probability distribution $p(\mathbf{y})$, or joint probability distribution $p(\mathbf{x}, \mathbf{y})$ if the data is split into an input and output part. Density estimation (Scott, 1992; Silverman, 1986) is concerned with extracting a model of this distribution from a given dataset.

In supervised learning, we are interested in the input-output relation, and we already saw in section 1.3.1 that the regression function is closely related to the conditional distribution $p(\mathbf{y}|\mathbf{x})$, which in turn can be calculated from the joint distribution $p(\mathbf{x}, \mathbf{y})$. In this way, the task of regression estimation can be viewed as a special case of density estimation. An algorithm which actually follows a density estimation approach to regression will be discussed in section 4.1. In an unsupervised setting, where the structure of the data is studied, an estimate of the density would also be valuable. Modes of the density, for example, could be identified with clusters.

Unfortunately, the amount of data which is required for a reasonable estimate of the density rises exponentially with its dimensionality (please recall the hyper-cube properties from Sec. 1.4). Thus, density estimation is considered to be practical only in low dimensionsal ($d \ll 10$) spaces.

1.5.2. Parametric vs. non-parametric methods

Besides the distinction between supervised and unsupervised learning, one can also coarsely divide learning algorithms into the two classes of *parametric* and *non-parametric*

approaches.

In parametric methods, the model which is fitted to the data has a specific and sometimes complex functional form. The whole flexibility of such a model lies in its parameters, which are adapted during learning. After the model is learned, it can be fully described by the function parameters, and also further usage of the model (e.g., for predicting the response to new input data) is usually quite efficient. An example of a parametric approach is the approximation of functions by polynomials up to a specific degree, where the adaptable parameters are the coefficients of the polynomial. As another example, in density estimation one might approximate the density function by a multivariate Gaussian distribution, where the parameters of the model are the mean vector and the covariance matrix.

In contrast, non-parametric methods rather directly use the training data to "build" the model. Simple functions and a few parameters may also be included, but the flexibility of the model lies in the data itself, which thus has to be memorized for further usage (e.g., prediction). As an example, consider the k-nearest-neighbor fit (e.g., Hastie et al., 2001, p. 14), which for a new input $\mathbf{x}$ first determines the k nearest neighbors $\mathbf{x}_i,\ i \in \mathcal{I}_\mathbf{x}$ from the training set, and then returns the mean of the corresponding outputs $\mathbf{y}_i,\ i \in \mathcal{I}_\mathbf{x}$. While the choice of k has some influence on the model (a larger k leads to less wriggly fits), the crucial part is the training data itself. As another example, kernel density estimation (Scott, 1992) approximates the density function by a superposition of simple kernel functions, each of which is centered at one data point. A bandwidth parameter determines the smoothness of the estimate, but the overall shape is given by the locations of the kernels.

Comparing the two approaches, parametric methods are usually more efficient (especially after training) and also easier to interpret, because of the pre-specified functional form. On the other hand, having to specify the functional form a priori can be viewed as a downside, because it may yield suboptimal results if the form does not match the data. As a function approximation example, periodic functions (a Fourier series) might yield better results than polynomials for one particular dataset, and vice versa for another.

Semi-parametric methods combine both approaches by systematically (depending on the data) superposing parametric models. An example for such a method is density estimation with mixture models (Bishop, 1995).

In the following, we will denote functions and their parameters as $\mathbf{f}(\mathbf{x}; \Theta)$, where Θ includes "non-parametric parameters", e.g., the bandwidth k from the k-nearest-neighbor fit as described above, but does *not* include hyper-parameters like the prefactor λ of a regularizing penalty or hyper-parameters that describe the function class, e.g., the maximum allowed degree of polynomials.

1.5.3. Maximum likelihood and Bayesian estimation

Briefly summarized, the maximum likelihood principle (Fisher, 1922) is based on the assumption that the distribution of observable data $\{\mathbf{z}\}$ can be parameterized and described by a density function $p(\mathbf{z}; \Theta)$. Under this assumption, the parameters Θ are adjusted such that the observation of a dataset $\mathcal{D} = \{\mathbf{z}_i \mid i = 1 \dots N\}$ becomes maxi-

mally probable. In particular, one maximizes the *likelihood*

$$L(\Theta) = \prod_{i=1}^{N} p(\mathbf{z}_i; \Theta), \tag{1.15}$$

or equivalently minimizes the negative log-likelihood

$$-\log L(\Theta) = -\sum_{i=1}^{N} \log p(\mathbf{z}_i; \Theta). \tag{1.16}$$

For a regression problem, the observed data are pairs $\mathbf{z}_i = (\mathbf{y}_i, \mathbf{x}_i)$, which are assumed to stem from a model

$$\mathbf{y} = \mathbf{f}(\mathbf{x}; \Theta) + \mathbf{u} \ , \quad \langle \mathbf{u} \rangle = 0 \ , \quad \mathbf{u} \sim p_{ns}(\mathbf{u}). \tag{1.17}$$

where $\mathbf{u}$ denotes zero-mean noise with a distribution $p_{ns}(\mathbf{u})$. We can factor the joint distribution as

$$p(\mathbf{z}_i; \Theta) = p(\mathbf{x}_i, \mathbf{y}_i; \Theta) = p(\mathbf{y}_i | \mathbf{x}_i; \Theta) p(\mathbf{x}_i), \tag{1.18}$$

where the marginal distribution $p(\mathbf{x})$ does not depend on the model parameters Θ. The conditional distribution $p(\mathbf{y}_i | \mathbf{x}_i; \Theta)$, however, does depend on Θ and can be expressed by

$$p(\mathbf{y}_i | \mathbf{x}_i; \Theta) = p_{ns} \left(\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i; \Theta) \right). \tag{1.19}$$

Inserting (1.19) into (1.16) yields

$$-\log L(\Theta) = -\sum_{i=1}^{N} \log p_{ns}(\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i; \Theta)) - C, \tag{1.20}$$

where $C = \sum_{i=1}^{N} \log p(\mathbf{x}_i)$ is independent of Θ. If we further assume that the noise distribution is isotropic Gaussian, i.e. $p_{ns}(\mathbf{u}) \propto \exp[-\frac{1}{2\sigma^2} \|\mathbf{u}\|^2]$, the negative log-likelihood becomes

$$-\log L(\Theta) = \frac{1}{2\sigma^2} \sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i; \Theta)\|^2 - C', \tag{1.21}$$

where C' holds terms indepent of Θ. Ignoring the constant C', this equation is proportional to (1.9), which means that maximizing the likelihood under the assumption of isotropic Gaussian noise is equivalent to minimizing the empirical risk as measured by the squared Euclidean error. By using $-\log p(\mathbf{x}_i, \mathbf{y}_i; \Theta)$ as a loss function $L(\mathbf{y}_i, \mathbf{f}(\mathbf{x}_i; \Theta))$, maximum likelihood estimation can be treated as a special case of empirical risk minimization (cf. Sec. 1.3.1 and 1.3.2).

In *Bayesian estimation*, the maximum likelihood principle is extended by viewing also the parameters Θ as random variables and by incorporating a prior distribution over the space of Θ. Using Bayes's theorem, the posterior distribution of the parameters given the data can be calculated as

$$p(\Theta; \mathcal{D}) = \frac{p(\mathcal{D}; \Theta) p(\Theta)}{p(\mathcal{D})}. \tag{1.22}$$

The most probable parameters Θ^* are retrieved by maximizing (1.22), where the prior distribution $p(\Theta)$ can be viewed as a regularization term. The "weight decay" approach for training neural networks is a well-known example for this relation, where a Gaussian prior over the space of the network's weights corresponds to penalizing large weights by their Euclidean norm (Bishop, 1995, p. 390).

1.5.4. The "kernel trick"

Any algorithm which can be cast in a form that only uses dot products between data vectors, but not the data vectors itself, can be transformed into a possibly much more powerful variant by use of the "kernel trick" (Aizerman, Braverman, and Rozonoer, 1964; Schölkopf and Smola, 2002). This transformation consists of the replacement

$$\mathbf{y}_i \cdot \mathbf{y}_j \rightarrow k(\mathbf{y}_i, \mathbf{y}_j), \tag{1.23}$$

where $k(\cdot, \cdot)$ is a symmetric positive semidefinite *kernel function*, that is

$$k(\mathbf{y}_i, \mathbf{y}_j) = k(\mathbf{y}_j, \mathbf{y}_i) \quad \text{and} \quad \sum_{i=1}^{N} \sum_{j=1}^{N} k(\mathbf{y}_i, \mathbf{y}_j) c_i c_j \geq 0 \tag{1.24}$$

for any finite set $\{\mathbf{y}_i \mid i = 1 \ldots N\}$ and any real[2] numbers c_i. Alternatively, one can state that the *kernel matrix* or *Gram matrix*

$$(\mathbf{K})_{ij} = k(\mathbf{y}_i, \mathbf{y}_j) \tag{1.25}$$

has to be symmetric and positive semidefinite. Fundamental research on such kernel functions has been carried out by Mercer (1909). Any "Mercer kernel" can be written in terms of a mapping $\Phi(\cdot)$ into a *feature space* $\mathcal{F}$, such that

$$k(\mathbf{y}_i, \mathbf{y}_j) = \langle \Phi(\mathbf{y}_i), \Phi(\mathbf{y}_j) \rangle, \tag{1.26}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product in that feature space. In effect, applying the kernel trick thus implicitly replaces the data vectors $\mathbf{y}_i$ by their feature space images $\Phi(\mathbf{y}_i)$. Depending on the choice of the kernel $k(\cdot, \cdot)$, the feature map $\Phi(\cdot)$ might involve complex nonlinearities, and $\mathcal{F}$ might be high- or even infinite-dimensional.

As an example, consider the homogenous polynomial kernel of order p

$$k(\mathbf{y}, \mathbf{y}') = (\mathbf{y} \cdot \mathbf{y}')^p = (\mathbf{y}^T \mathbf{y}')^p. \tag{1.27}$$

For $p = 2$ and two-dimensional data vectors $\mathbf{y} = (y_1, y_2)^T$ and $\mathbf{y}' = (z_1, z_2)^T$, we can write

$$\begin{aligned} k(\mathbf{y}, \mathbf{y}') &= (y_1, y_2)(z_1, z_2)^T \\ &= (y_1 z_1 + y_2 z_2)^2 \\ &= (y_1^2 z_1^2 + y_1 z_1 y_2 z_2 + y_2^2 z_2^2) \\ &= (y_1^2, y_1 y_2, y_2^2)(z_1^2, z_1 z_2, z_2^2)^T, \end{aligned} \tag{1.28}$$

[2]For simplicity, this discussion is restricted to real-valued kernels.

by which the feature map can be identified as

$$\Phi(\mathbf{y}) = (y_1^2, y_1 y_2, y_2^2)^T. \tag{1.29}$$

In general, the polynomial kernel (1.27) implies a mapping of a data vector on another vector which contains all monomials of order p, that is, all possible products of p elements of the original data vector. Since there are

$$\begin{pmatrix} p+d-1 \\ p \end{pmatrix} = \frac{(p+d-1)!}{(d-1)!p!} \tag{1.30}$$

different monomials for data vectors of length d, the dimensionality of the feature space rises very quickly with p (Schölkopf and Smola, 2002). The computational effort, however, is hardly influenced as long as only the kernel function has to be computed.

All products of order p or lower can be implicitly calculated by the inhomogenous polynomial kernel

$$k(\mathbf{y}, \mathbf{y}') = (\mathbf{y}^T \mathbf{y}' + c)^p, \tag{1.31}$$

where the constant $c > 0$ determines the influence of lower-order products. Another popular example of a Mercer kernel is the Gaussian kernel

$$k(\mathbf{y}, \mathbf{y}') = \exp(-\frac{1}{2\sigma^2}\|\mathbf{y} - \mathbf{y}'\|^2), \tag{1.32}$$

which yields an implicit mapping into an infinite-dimensional feature space (Schölkopf and Smola, 2002).

The application of Mercer kernels in machine learning algorithms has become very popular since the introduction of "kernelized" support vector machines for classification (Boser, Guyon, and Vapnik, 1992) and regression (Drucker, Burges, Kaufman, Smola, and Vapnik, 1997).

2. Related Methods

2.1. Vector quantization and clustering

Although not directly related to manifold learning at first sight, vector quantization and clustering are important ingredients of some of the algorithms that will be described later on. This section will introduce the basic ideas and review selected methods.

The goal of vector quantization is to represent a set of patterns or data vectors $\{\mathbf{y}_i \in \mathbb{R}^d,\ i = 1 \ldots N\}$ by a smaller set of prototypes or *code-book* vectors $\mathcal{C} = \{\mathbf{c}_j \in \mathbb{R}^d,\ j = 1 \ldots K\},\ K < N$. Such a representation can be used for lossy compression: instead of storing or transmitting all $d \cdot N$ values of the original dataset, one stores only the code-book $\mathcal{C}$ ($d \cdot K$ values) and further the index of each data vector's closest code-book vector (N integer values). Thus, if we restrict ourselves to the Euclidean distance, a data vector $\mathbf{y}$ is represented by the index

$$j^* = g(\mathbf{y}; \mathcal{C}) = \arg\min_j \|\mathbf{y} - \mathbf{c}_j\|^2, \tag{2.1}$$

and the *mean square quantization error*, that is, the average error when reconstructing the data from their corresponding code-book vectors, is given by

$$E = \frac{1}{N} \sum_{i=1}^{N} \min_j \|\mathbf{y}_i - \mathbf{c}_j\|^2. \tag{2.2}$$

By introducing a discrete latent variable $x \in \{1 \ldots K\}$ with realizations $x_i = g(\mathbf{y}_i; \mathcal{C})$ and a mapping $\mathbf{f}(x; \mathcal{C}) = \mathbf{c}_x$, this error can also be written as

$$E = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{f}(x_i, \mathcal{C})\|^2, \tag{2.3}$$

which very closely resembles (1.14). In this way, vector quantization can be treated as a generalized regression problem (Meinicke, 2000).

The definition of $g(\cdot)$ in (2.1) suggests to view vector quantization also as a *hard clustering* algorithm, because $g(\cdot)$ effectively partitions the data space into disjoint patches, the so-called *Voronoi cells* (cf. Fig. 2.1)

$$\begin{aligned} \mathcal{V}_j &= \{\mathbf{y} \mid g(\mathbf{y}; \mathcal{C}) = j\} \\ &= \{\mathbf{y} \mid \forall_{i \neq j} \|\mathbf{y} - \mathbf{c}_j\|^2 < \|\mathbf{y} - \mathbf{c}_i\|^2\}\ ,\quad i, j \in \{1 \ldots n\}. \end{aligned} \tag{2.4}$$

The index j can be thought of as a cluster label, and the code-book vector $\mathbf{c}_j$ can thus also be called "cluster center".

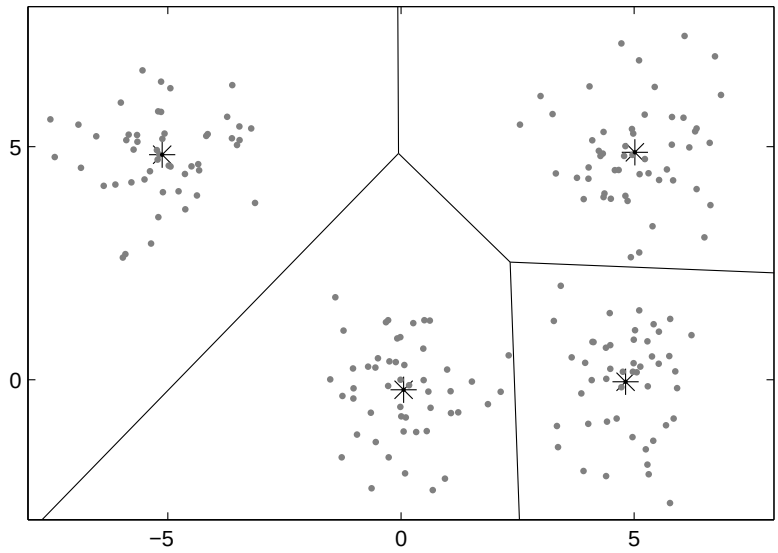


Figure 2.1: Voronoi tessellation induced by 4 code-book vectors (black stars), which were fitted to 200 data points (grey dots) by the K-Means algorithm.

2.1.1. K-Means clustering

The K-Means algorithm (MacQueen, 1967) is a very simple and popular hard clustering method. In a vector quantization context, it is also known as the generalized Lloyd algorithm. The method can be summarized as follows:

1. Choose the desired number K of clusters.
2. Initialize the K code-book vectors $\mathbf{c}_j,\ j = 1 \dots K$ randomly, for example by setting them to randomly selected data vectors.
3. Assign each data vector $\mathbf{y}_i$ to its closest code-book vector by setting
$$x_i = \arg\min_j \|\mathbf{y} - \mathbf{c}_j\|^2.$$
4. Update each code-book vector by setting it to the mean of its associated data vectors
$$\mathbf{c}_j = \frac{\sum_{i=1}^{N} \delta_{jx_i} \mathbf{y}_i}{\sum_{l=1}^{N} \delta_{jx_l}}. \tag{2.5}$$
5. Iterate steps 3 and 4 until convergence, that is, until the assignments x_i do not change anymore. Alternatively, stop the algorithm if the relative change of the quantization error is below a certain threshold.

Step 3 minimizes the quantization error (2.2) with respect to the assignments x_i, while step 4 minimizes that error with respect to the code-book vectors. Both steps can only improve the current solution or leave it unchanged, which is why in many practical situations the algorithm converges very quickly. Unfortunately, it is well known that

the K-Means algorithm often gets trapped in bad local minima of the highly non-convex objective function (2.2). The commonly used strategy to avoid such suboptimal solutions is to just compare the results of multiple runs with different initializations.

2.1.2. Clustering with mixture models

Bishop (1995) describes a maximum likelihood approach to clustering, where he approximates the distribution of the data $\mathbf{y}_i$ by a mixture model

$$p(\mathbf{y}_i) = \sum_{k=1}^{K} p(\mathbf{y}_i|k)P(k). \tag{2.6}$$

Here, $P(k)$ is the *prior* probability that any given data vector stems from the k-th cluster, and $p(\mathbf{y}_i|k)$ describes the distribution of the data that belongs to that cluster, which is assumed to be isotropic Gaussian with mean $\mathbf{c}_k$ and covariance $\sigma_k^2\mathbf{I}_d$, that is

$$p(\mathbf{y}_i|k) = \frac{1}{(2\pi\sigma_k^2)^{\frac{d}{2}}} \exp\left(-\frac{1}{2\sigma_k^2}\|\mathbf{y}_i - \mathbf{c}_k\|^2\right). \tag{2.7}$$

By introducing latent variables $x_i \in \{1 \dots K\}$, which hypothetically indicate to which cluster a data vector $\mathbf{y}_i$ belongs, the task of finding the best parameters $\Theta = \{\mathbf{c}_k, \sigma_k \mid k = 1 \dots K\}$ can be treated as a "maximum likelihood from incomplete data" problem.

A general strategy for solving such problems is given by the *expectation-maximization* (EM) algorithm (Dempster, Laird, and Rubin, 1977), which consists of two alternating steps. In the E-step, given current estimates Θ^0 and $P^0(k)$ of the parameters and the prior probabilities, respectively, one calculates the posterior probabilities or responsibilities

$$r_{ik}^0 = P(x_i = k|\mathbf{y}_i; \Theta^0) = \frac{p(\mathbf{y}_i|k; \Theta^0)P^0(k)}{p(\mathbf{y}_i; \Theta^0)}. \tag{2.8}$$

From these probabilities, one can calculate the expected log-likelihood of the complete data (that is, including the variables x_i)

$$\left\langle L^{comp}(\Theta)\right\rangle_x = \sum_{x_1=1}^{K}\sum_{x_2=1}^{K}\cdots\sum_{x_N=1}^{K}\prod_{i=1}^{N} P(x_i|\mathbf{y}_i; \Theta^0)L^{comp}(\Theta), \tag{2.9}$$

where $L^{comp}(\Theta)$ is viewed as a function of new parameters Θ and given by

$$L^{comp}(\Theta) = \log\prod_{i=1}^{N} p(x_i, \mathbf{y}_i; \Theta) = \sum_{i=1}^{N} \log\Big(p(\mathbf{y}_i|x_i; \Theta)P(x_i)\Big). \tag{2.10}$$

Bishop (1995) shows that (2.9) can more compactly be written as

$$\left\langle L^{comp}(\Theta)\right\rangle_x = \sum_{k=1}^{K}\sum_{i=1}^{N} r_{ik}^0 \log\Big(p(\mathbf{y}_i|k; \Theta)P(k)\Big). \tag{2.11}$$

In the M-step, the expected log-likelihood (2.9) is maximized with respect to the parameters Θ and the prior probabilities $P(k)$, which yields a new estimate of the mixture model.

The EM algorithm typically converges after few iterations, but unfortunately also suffers from getting stuck in bad local minima.

2.1.3. K-Harmonic Means

The K-Harmonic Means algorithm (Zhang, Hsu, and Dayal, 1999) has been proposed as an improvement over K-Means regarding the sensitivity towards bad initializations. Its main difference from K-Means is a modified objective function. Instead of minimizing the mean square quantization error (2.2), K-Harmonic Means aims at a minimal *harmonic average* of the distances between data vectors and cluster centers. This measure is given by

$$E_{har} = \frac{1}{N} \sum_{i=1}^{N} \frac{K}{\sum_{j=1}^{K} \frac{1}{\|\mathbf{y}_i - \mathbf{c}_j\|^2}}. \tag{2.12}$$

For minimizing (2.12), the authors propose the recursive update formula

$$\mathbf{c}_j = \left(\sum_{i=1}^{N} \frac{1}{d_{ij}^4 \left(\sum_{k=1}^{K} \frac{1}{d_{ik}^2} \right)^2} \mathbf{y}_i \right) \left(\sum_{i=1}^{N} \frac{1}{d_{ij}^4 \left(\sum_{k=1}^{K} \frac{1}{d_{ik}^2} \right)^2} \right)^{-1}, \tag{2.13}$$

where $d_{ij} = \|\mathbf{y}_i - \mathbf{c}_j\|$ are the distances between the data vectors and the code-book vectors from the last iteration. To simplify the update formula, one could introduce the "weights"

$$w_{ji} = \frac{1}{d_{ij}^4 \left(\sum_{k=1}^{K} \frac{1}{d_{ik}^2} \right)^2} = \frac{1}{\|\mathbf{y}_i - \mathbf{c}_j\|^4 \left(\sum_{k=1}^{K} \frac{1}{\|\mathbf{y}_i - \mathbf{c}_j\|^2} \right)^2}, \tag{2.14}$$

which determine how much a data vector $\mathbf{y}_i$ contributes to the calculation of a code-book vector $\mathbf{c}_j$ in the next step. Inserting the weights into (2.13) yields

$$\mathbf{c}_j = \frac{\sum_{i=1}^{N} w_{ji} \mathbf{y}_i}{\sum_{l=1}^{N} w_{jl}} \tag{2.15}$$

and reveals a direct correspondence between w_{ji} and the terms δ_{jx_i} as used in the K-Means update equation (2.5).

The K-Harmonic Means algorithm can be generalized by replacing the term $\|\mathbf{y}_i - \mathbf{c}_j\|^2$ in (2.12) by $\|\mathbf{y}_i - \mathbf{c}_j\|^p$ with $p > 2$ (Zhang, 2000). While the algorithm outperforms the standard K-Means already for $p = 2$, higher values (e.g. $p = 3.5$) yield even better results (with respect to the mean squared quantization error) and more robustness towards the choice of the initial code-book (Zhang, 2000; Hamerly and Elkan, 2002).

2.1.4. Further algorithms

Linde, Buzo, and Gray (1980) proposed a vector quantization algorithm that updates code-book vectors similar to K-Means, but includes a dynamic adaption of the parameter K by splitting up clusters.

Rose, Gurewitz, and Fox (1990) aimed at avoiding bad local minima by utilizing a deterministic annealing approach to minimize the quantization error. Instead of a hard assignment of data vectors to clusters, they assigned a vector $\mathbf{y}_i$ to the cluster j (represented by $\mathbf{c}_j$) with the probability

$$P(j|\mathbf{y}_i) = \frac{\exp(-\beta \|\mathbf{y}_i - \mathbf{c}_j\|^2)}{\sum_{k=1}^{K} \exp(-\beta \|\mathbf{y}_i - \mathbf{c}_k\|^2}, \tag{2.16}$$

which is known as the Gibbs distribution. By varying the parameter β, this assignment can be made completely fuzzy ($\beta = 0$, each data vector belongs equally to all clusters), hard ($\beta \to \infty$, each data vector belongs to only one cluster), or anything in between. The authors then propose an optimization scheme which iteratively determines the best cluster centers $\mathbf{c}_j$ for a series of increasing values of β.

A simple neural network model for vector quantization has been introduced by Kohonen (e.g., Kohonen, 2001). The algorithm is based on a set of "neurons", which are characterized by a weight vector $\mathbf{w}_i$, together with a "winner-takes-all" learning rule: When presenting a data vector $\mathbf{y}$ to the network, the neuron with the closest reference-vector $\mathbf{w}_*$ gets adapted by

$$\mathbf{w}_*^{new} = \mathbf{w}_*^{old} + \gamma(\mathbf{y} - \mathbf{w}_*^{old}), \tag{2.17}$$

which effectively shifts the weight towards the input vector $\mathbf{y}$. By identifying the weight vectors as code-book vectors, this VQ learning rule can be viewed as an online variant of the K-Means algorithm. In order to reach convergence, the learning rate γ has to be decreased over time.

Martinetz and Schulten (1991) proposed a "Neural-Gas" network for vector quantization, where the winner-takes-all rule is replaced by a competitive learning scheme. For each data vector $\mathbf{y}$ presented to the network, the neurons are ranked by their response to $\mathbf{y}$, i.e., the distance between $\mathbf{y}$ and their weights $\mathbf{w}_i$. Then, all neurons get adapted by

$$\mathbf{w}_i^{new} = \mathbf{w}_i^{old} + \gamma f(r_i)(\mathbf{y} - \mathbf{w}_i^{old}) \quad \forall i. \tag{2.18}$$

Here, $f(r_i)$ denotes a decreasing function of the rank, that is, the adaption is strongest for the closest neuron. During training, the shape of $f(\cdot)$ is modified such that far away neurons get adapted less strongly. This is similar in spirit to the annealing scheme of Rose et al. (1990), but yields superior quantization performance as reported by Martinetz, Berkovich, and Schulten (1993). The Neural-Gas algorithm also aims at representing the topology of the data, which is achieved by creating links between neurons with similar response.

2.2. Principal Component Analysis

Principal Component Analysis (PCA) is probably the most widely used technique for dimension reduction. Early work was carried out by Pearson (1901), who fitted lines and planes to a given set of points, and by Hotelling (1933), who was concerned with the analysis of multivariate random variables.

The standard textbook variant of PCA (see e.g. Jolliffe, 2002) can be described as follows: Given a set of data vectors $\mathbf{y}_i$, $i = 1 \ldots N$ in a d-dimensional space, the goal of PCA is to find the affine subspace (or linear manifold) of dimension $q < d$ which captures as much variance of the data as possible. It can be shown that this subspace passes through the mean

$$\bar{\mathbf{y}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{y}_i \tag{2.19}$$

of the data and is spanned by the eigenvectors corresponding to the q largest eigenvalues of the sample *covariance matrix*

$$\mathbf{C} = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{y}_i - \bar{\mathbf{y}})(\mathbf{y}_i - \bar{\mathbf{y}})^T. \tag{2.20}$$

Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$ be the eigenvalues of $\mathbf{C}$ and $\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_d$ the corresponding (orthonormal) eigenvectors, we can define a $d \times q$-matrix

$$\mathbf{V}_q = (\hat{\mathbf{v}}_1, \hat{\mathbf{v}}_2, \ldots, \hat{\mathbf{v}}_q) \tag{2.21}$$

and an affine mapping

$$\mathbf{x}_i = \mathbf{g}(\mathbf{y}_i) = \mathbf{V}_q^T(\mathbf{y}_i - \bar{\mathbf{y}}) \tag{2.22}$$

from the data space into the q-dimensional space of the *principal components*[1] $\mathbf{x}_i$, as well as another mapping

$$\tilde{\mathbf{y}}_i = \mathbf{f}(\mathbf{x}_i) = \mathbf{V}_q\mathbf{x}_i + \bar{\mathbf{y}} \tag{2.23}$$

back to the d-dimensional data space, with $\tilde{\mathbf{y}}_i$ being the projections of the data vectors $\mathbf{y}_i$ onto the linear manifold. By this construction,

1. the principal components $\mathbf{x}_i$ have zero mean.

2. the principal components are uncorrelated, that is

$$\sum_{i=1}^{N} x_{ji}x_{ki} = 0 \quad \forall j \neq k. \tag{2.24}$$

3. the variance of the data along the axis $\hat{\mathbf{v}}_k,\ k = 1 \ldots q$ is given by

$$\mathrm{var}(\hat{\mathbf{v}}_k^T\mathbf{y}) = \mathrm{var}(\hat{\mathbf{e}}_k^T\mathbf{x}) = \lambda_k \tag{2.25}$$

   where $\hat{\mathbf{e}}_i$ denotes the k-th standard basis vector in $\mathbb{R}^q$.

An important property of PCA is that one can judge how many principal components (the value of q) one needs to extract *after* running the main part of the algorithm. Since the spectrum of $\mathbf{C}$ equals the variance along the different axes (2.25), one can look for "gaps" in the spectrum, which often indicate a division between meaningful axes and axes containing mostly noise (cf. the left plot of Fig. 2.2). Alternatively, one can compare the sum of the largest q eigenvalues to the sum of all eigenvalues and set q in a way that e.g. 95% of the variance is captured.

In the following, an alternative formulation of PCA is presented, which better matches our scope of manifold learning: We now wish to reconstruct the data vectors $\mathbf{y}_i$ from their low-dimensional representation as indicated in (1.13). If we restrict the function class to affine mappings

$$\mathbf{f}(\mathbf{x}; \mathbf{A}, \mathbf{b}) = \mathbf{A}\mathbf{x} + \mathbf{b} \tag{2.26}$$

[1] It should be noted that sometimes the eigenvectors $\hat{\mathbf{v}}_k$ are called "principal components".

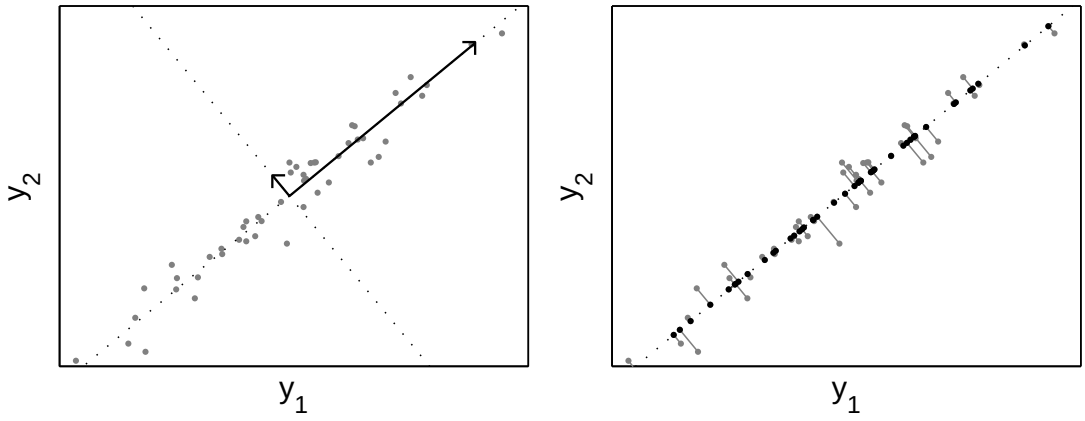


Figure 2.2: An example of applying PCA to 2-D data. The dataset consists of 100 points (gray dots), which are distributed along a straight line. The left plot shows the two principal axes (dashed lines) as well as the eigenvectors of the covariance matrix (black arrows). For visualizing the importance of the axes, the eigenvectors $\hat{\mathbf{v}}_i$ have been scaled by $2\sqrt{\lambda_i},\ i = 1, 2$. The right plot shows the projection (black dots) of the dataset onto the first principal axis (dashed line).

with $\mathbf{A} \in \mathbb{R}^{d \times q}$ and $\mathbf{b} \in \mathbb{R}^d$, the empirical reconstruction error (1.14) can be written as a function of the latent variables and the parameters $\mathbf{A}$ and $\mathbf{b}$, i.e.

$$R(\{\mathbf{x}_i\}, \mathbf{A}, \mathbf{b}) = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{A}\mathbf{x}_i - \mathbf{b}\|^2. \tag{2.27}$$

Minimizing (2.27) without further restrictions is an ill-posed problem, because we did not fixate the function model enough[2]. If we require the latent variables to have zero mean and the matrix $\mathbf{A}$ to have orthonormal columns, we retrieve (see e.g. C. R. Rao, 1964)

$$\mathbf{A} = \mathbf{V}_q\ , \quad \mathbf{b} = \bar{\mathbf{y}} \ \text{ and } \ \mathbf{x}_i = \mathbf{V}_q^T(\mathbf{y}_i - \bar{\mathbf{y}}), \tag{2.28}$$

that is, the same solution as for the variance-based approach.

The main drawback of PCA is its restriction to linear manifolds. As an example, Fig. 2.3 shows the results of PCA on a 2-D dataset which consists of 100 points distributed along a half circle. While the data is essentially one-dimensional, a PCA finds considerable variance along both principal axes. Consequently, a projection of the data onto the first principal axis (the reduction from 2-D to 1-D) yields a much larger error than a projection onto the underlying half circle, which is depicted in Fig. 2.4.

If the training data is not given as a whole set but rather sequentially, or if a PCA model should be updated when new training data arrives, replacing the eigendecomposition by an online learning scheme can be advantageous. For this, E. Oja (1982) introduced a neural model for extracting the first principal eigenvector, which is based on the modified Hebbian learning rule

$$\mathbf{w}^{new} = \mathbf{w}^{old} + \gamma h(\mathbf{y} - h\mathbf{w}^{old}). \tag{2.29}$$

[2] We could set any value for $\mathbf{b}$ by a corresponding adjustment of the mean of the latent variables. Similarly, any non-singular transformation $\mathbf{A}' = \mathbf{A}\mathbf{T}$ can be accounted for by left-multiplying the latent variables with $\mathbf{T}^{-1}$.

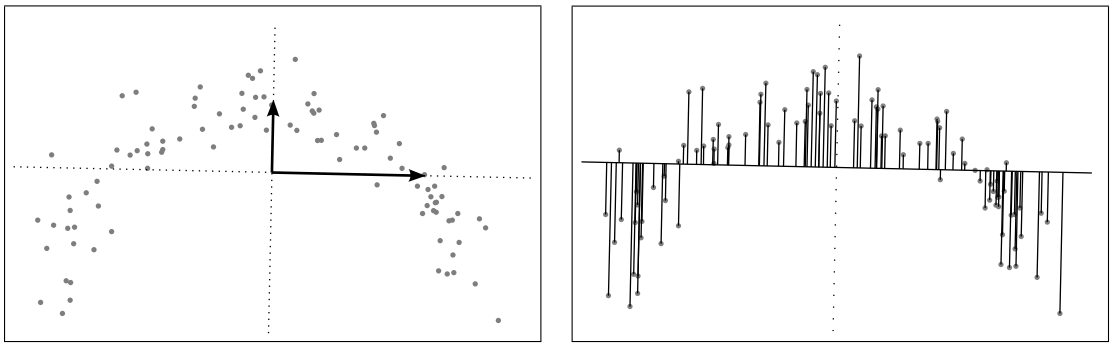


Figure 2.3: Another example of applying PCA to 2-D data. The dataset consists of 100 points (gray dots), which are distributed along a half circle. The left plot shows the two principal axes (dashed lines) as well as the eigenvectors of the covariance matrix (black arrows, scaled as in Fig. 2.2). The right plot shows the projection (along the black lines) of the dataset onto the first principal axis.

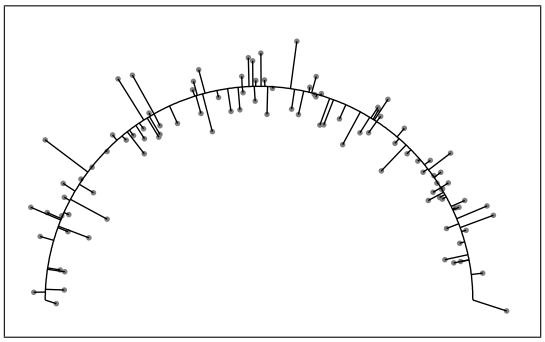


Figure 2.4: Projection of the half circle dataset (cf. Fig. 2.3) onto the true underlying manifold.

Here, γ is a learning rate parameter, and $h = \mathbf{y}^T \mathbf{w}^{old}$ is the output of a single linear neuron with weight vector $\mathbf{w}$, which converges to the desired eigenvector. Sanger (1989) proposed a "Generalized Hebbian Algorithm" for a network model that can extract multiple principal components. A recent overview of such PCA learning algorithms has been given by Möller and Könies (2004).

2.2.1. Probabilistic PCA

Tipping and Bishop (1997, 1999b) derived a probabilistic variant of PCA, which also links PCA to *factor analysis* (Everitt, 1984; Bartholomew and Knott, 1999). Factor analysis aims at relating an observable d-dimensional random variable $\mathbf{y}$ to a q-dimensional latent random variable $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_q)$. Its model is given by

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{m} + \mathbf{u}, \tag{2.30}$$

with parameters $\mathbf{W} \in \mathbb{R}^{d \times q}$ and $\mathbf{m} \in \mathbb{R}^d$, and $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \Psi)$ denoting zero-mean Gaussian noise with a diagonal covariance matrix Ψ. The model of probabilistic PCA differs only in the restriction $\Psi = \sigma^2 \mathbf{I}_d$. Then, the marginal distribution of $\mathbf{y}$ is also

Gaussian and given by

$$\mathbf{y} \sim \mathcal{N}(\mathbf{m}, \mathbf{W}\mathbf{W}^T + \sigma^2 \mathbf{I}_d). \tag{2.31}$$

Maximizing the likelihood of observing a dataset $\{\mathbf{y}_i \mid i = 1 \dots N\}$ given this model, one retrieves

$$\mathbf{W}_{ML} = \mathbf{V}_q (\mathbf{\Lambda}_q - \sigma^2_{ML} \mathbf{I}_q)^{\frac{1}{2}} \mathbf{R}, \tag{2.32}$$

where $\mathbf{R}$ is an arbitrary orthogonal $q \times q$ matrix, $\mathbf{\Lambda}_q = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_q)$ contains the q largest eigenvalues of the data's covariance matrix (2.20), and $\mathbf{V}_q$ is given by (2.21), that is, probabilistic PCA finds the same principal axes as standard PCA. The optimal noise variance is given by

$$\sigma^2_{ML} = \frac{1}{d-q} \sum_{i=q+1}^{d} \lambda_i, \tag{2.33}$$

which is the average variance along the "lost" axes. While similar in result to standard PCA, the probabilistic approach has some advantages. For example, it can be utilized to handle the case where not all data vectors $\mathbf{y}_i$ are complete (Tipping and Bishop, 1999b), and it can be extended to mixtures of PCA models (see the next section) in a natural way.

2.2.2. Local PCA and mixture models

As introduced by Kambhatla and Leen (1994), a simple and fast approach to overcome the limitations of global PCA is to first partition the dataset into clusters, e.g. utilizing the K-Means algorithm. Then, PCA is carried out within each Voronoi cell, yielding multiple local linear models. Kambhatla and Leen (1994) themselves stated that this solution is not optimal, because the clustering step ignores how well a local linear model can reconstruct the data within its Voronoi cell, and conceived the following refinement: Within the K-Means iterations, the data vectors are not assigned to the nearest codebook vector $\mathbf{c}_j$, but to the PCA model which can reconstruct the data vector best.

Figure 2.5 shows the result of fitting 4 local PCA models to the "half circle" example dataset. Please note how well each point is reconstructed by its closest PCA model (indicated by the black projection lines), but also how far away data points get assigned to the same PCA model.

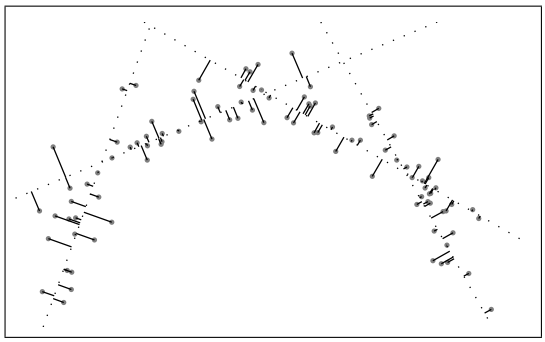


Figure 2.5: A combination of four local PCA models (indicated by dotted lines) can yield a much smaller projection error than a global PCA model (cf. Fig. 2.3).

Bregler and Omohundro (1994) applied a similar combination of K-Means clustering and local PCA in a lipreading application. They blended the local PCA models, denoted by $\mathbf{P}_k(\mathbf{y})$, by help of the formula

$$\mathbf{P}(\mathbf{y}) = \sum_{k=1}^{K} \frac{G_k(\mathbf{y})}{\sum_{j=1}^{K} G_j(\mathbf{y})} \mathbf{P}_k(\mathbf{y}), \tag{2.34}$$

where $G_k(\cdot)$ denotes a Gaussian distribution which is centered at the code-book vector $\mathbf{c}_k$, with "a variance determined by the local sample density" (Bregler and Omohundro, 1994). Zhang, Fu, and Yan (1998) used the Neural-Gas algorithm in conjunction with PCA to fit multiple local linear models to a dataset of handwritten digits, whereas Hinton, Dayan, and Revow (1997) tackled that problem by combining local PCA models with a soft-clustering approach based on the EM algorithm.

As Tipping and Bishop (1999a) point out, all aforementioned algorithms are based on a two-stage approach (clustering and PCA), aim at optimizing a global error criterion (the reconstruction error), and do not yield a model of the data distribution. As an alternative, the probabilistic PCA formulation can be extended to a mixture model by setting (Tipping and Bishop, 1999a)

$$p(\mathbf{y}) = \sum_{i=1}^{K} \pi_i p(\mathbf{y}|i), \tag{2.35}$$

where the π_i are mixture coefficients with $\sum_i \pi_i = 1$, and $p(\mathbf{y}|i)$ are Gaussian distributions as given by (2.31), each with a different set of parameters $\mathbf{W}_i, \mathbf{m}_i$ and σ_i. The optimal values of these parameters for a given dataset $\{\mathbf{y}_i\}$ can be retrieved by maximizing the likelihood via the EM algorithm (cf. Sec. 2.1.2).

With the aforementioned density model, it is unlikely that data points get assigned to a far away (regarding its center) local model, even if they lie along the principal axes of that model. This feature is shared by a recently introduced algorithmic combination of Neural-Gas and local PCA (Möller and Hoffmann, 2004), where the Mahalanobis distance (Mahalanobis, 1957) is used for the (soft) assignment of data vectors to local models.

While a combination of local PCA models can efficiently yield a data space representation of an embedded manifold, it does not provide a coherent set of lower dimensional coordinates $\mathbf{x}_i$, as in the case of a global PCA model. Rather the latent variables are pairs which contain the assignment to the local models, and the low-dimensional positions within that models.

2.2.3. Kernel PCA

Schölkopf, Smola, and Müller (1998) derived a nonlinear variant of global PCA by applying the "kernel trick" (cf. Sec. 1.5.4), which requires to first write the algorithm in dot product form. For simplicity, we assume that our dataset has zero mean, that is $\frac{1}{N}\sum_{i=1}^{N} \mathbf{y}_i = \mathbf{0}$. The eigenvalue equation for the sample covariance matrix is then given by

$$\lambda \mathbf{v} = \mathbf{C}\mathbf{v} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{y}_i \mathbf{y}_i^T \mathbf{v}. \tag{2.36}$$

The eigenvectors can only lie in the span of the data, which has two consequences. Firstly, we can express any eigenvector as a linear combination $\mathbf{v} = \sum_{j=1}^{N} a_j \mathbf{y}_j$, resulting in

$$\lambda \sum_{j=1}^{N} a_j \mathbf{y}_j = \frac{1}{N} \sum_{i,j=1}^{N} a_j \mathbf{y}_i \mathbf{y}_i^T \mathbf{y}_j. \tag{2.37}$$

Secondly, we can retrieve a set of N equations which are equivalent to (2.36) by left-multiplying (2.37) with $\mathbf{y}_l^T, l = 1 \dots N$, which yields

$$\lambda \sum_{j=1}^{N} a_j \left(\mathbf{y}_l^T \mathbf{y}_j\right) = \frac{1}{N} \sum_{i,j=1}^{N} a_j \left(\mathbf{y}_l^T \mathbf{y}_i\right) \left(\mathbf{y}_i^T \mathbf{y}_j\right) \ , \quad l = 1 \dots N. \tag{2.38}$$

If we introduce the $N \times N$ Gram matrix $(\mathbf{K})_{ij} = \mathbf{y}_i^T \mathbf{y}_j$ and the vector notation $(\mathbf{a})_i = a_i$, we can write (2.38) as

$$\lambda \mathbf{K}\mathbf{a} = \frac{1}{N} \mathbf{K}\mathbf{K}\mathbf{a}, \tag{2.39}$$

which is solved when $(N\lambda, \mathbf{a})$ is an eigenvalue/eigenvector pair of $\mathbf{K}$. In order to normalize the original eigenvector $\mathbf{v}$ to unit length, we can calculate

$$1 = \|\mathbf{v}\|^2 = \mathbf{v}^T \mathbf{v} = \sum_{i,j=1}^{N} a_j \mathbf{y}_j^T \mathbf{y}_i a_i = \mathbf{a}^T \mathbf{K}\mathbf{a} = \lambda N \mathbf{a}^T \mathbf{a} \tag{2.40}$$

and scale $\mathbf{a}$ accordingly. The projection of $\mathbf{y}_i$ onto e.g. the first principal axis $\mathbf{v}_1$ can be calculated by

$$x_i^1 = \mathbf{v}_1^T \mathbf{y}_i = \sum_{j=1}^{N} a_j^1 \mathbf{y}_j^T \mathbf{y}_i = (\mathbf{K}\mathbf{a}^1)_i = \lambda_1 N a_i^1, \tag{2.41}$$

where $\mathbf{a}^1$ is the first (properly normalized) eigenvector of $\mathbf{K}$.

In this form, PCA is carried out by decomposing the $N \times N$ Gram matrix, which is actually favorable in the case $d > N$. More importantly, the dot product $\mathbf{y}_i^T \mathbf{y}_j$ can now be replaced by a Mercer kernel $k(\mathbf{y}_i, \mathbf{y}_j)$, and thus PCA is implicitly carried out in the corresponding feature space. Centering the data in that space is not as simple as in the data space, but it can also be expressed in terms of dot products. For the details on this step, we refer to the work of Schölkopf et al. (1998).

Kernel PCA is essentially a two-stage algorithm which first implicitly maps the data into a feature space, and which afterwards searches for the axes of maximal variance within that space, in the hope that the nonlinear mapping has "stretched out" the data onto a linear manifold. Like standard PCA, kernel PCA does not suffer from the existence of local minima despite its ability to detect nonlinear features. The difficulty lies rather in the choice of the Mercer kernel and its parameters.

Figure 2.6 shows the results of using kernel PCA on the half circle dataset, utilizing both a polynomial and a Gaussian kernel. Please note that in contrast to standard PCA, kernel PCA can actually extract more features (find more axes) than the dimensionality of the original data.

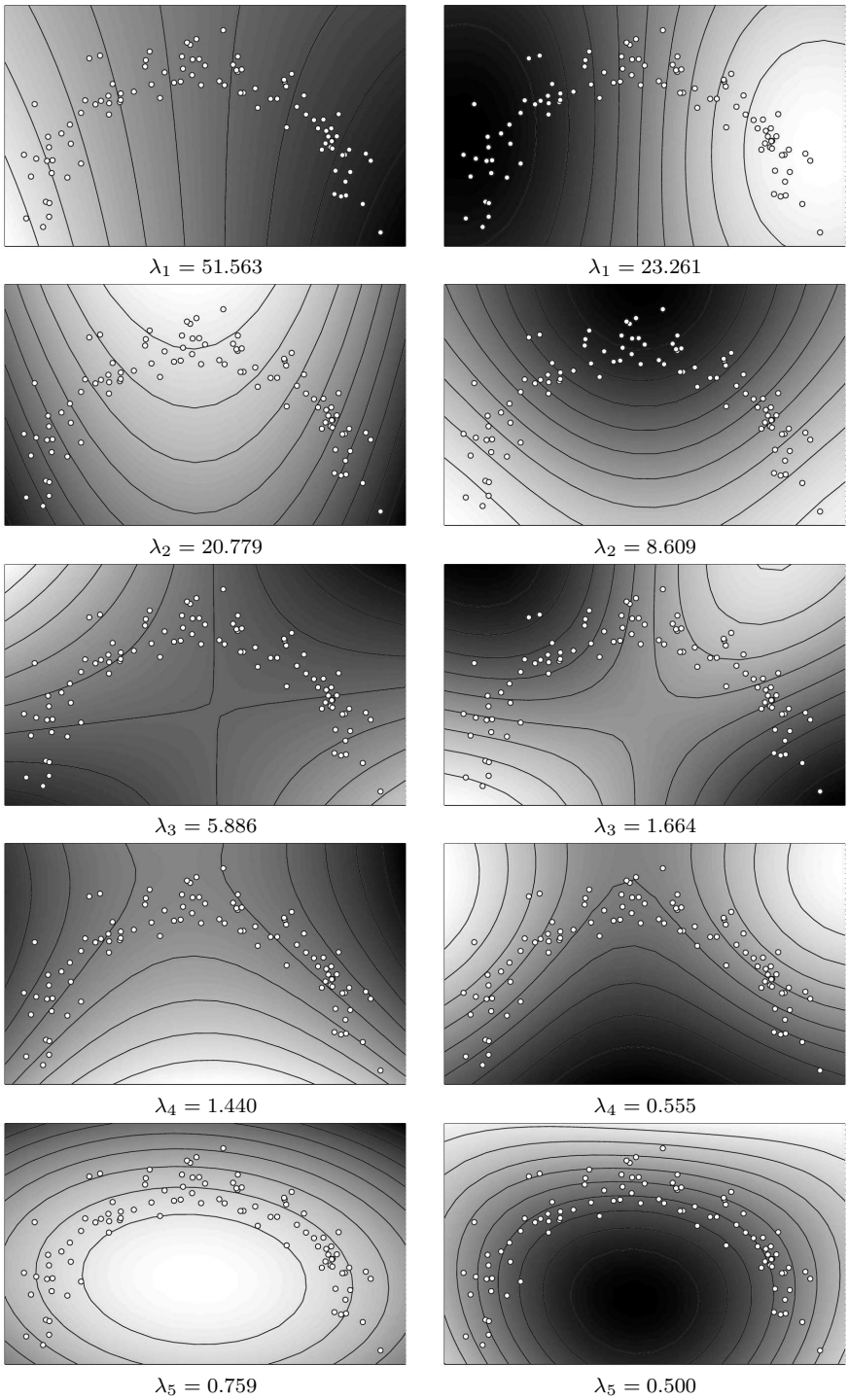


Figure 2.6: Kernel PCA results for the half circle dataset (white dots). The contour lines are perpendicular to the feature space eigenvectors. The corresponding eigenvalues are depicted below the plots. Left: polynomial kernel $k(\mathbf{y}, \mathbf{y}') = (\mathbf{y}^T\mathbf{y}' + 0.5)^2$, right: Gaussian kernel $k(\mathbf{y}, \mathbf{y}') = \exp(-0.5\|\mathbf{y} - \mathbf{y}'\|^2)$.

2.3. Auto-associative neural networks

The multi-layer perceptron (MLP) is a very popular kind of neural network (see e.g. Hertz et al., 1991; Bishop, 1995). It consists of multiple layers of neurons and processes the input data in a feed-forward fashion: an input vector $\mathbf{x}$ (layer 0) is multiplied by weight vectors $\mathbf{w}_i^1$ which yields the synaptic summation of the neurons n_i^1 in layer 1. These neurons might be equipped with a nonlinear activation function $\sigma(\cdot)$, and thus yield output values $\sigma(\mathbf{w}_i^1 \cdot \mathbf{x} - b_i^1)$, where b_i^1 is an optional "bias term". The outputs of layer 1 are then treated as inputs for layer 2, and so on. An example of an MLP is depicted in Fig. 2.7.

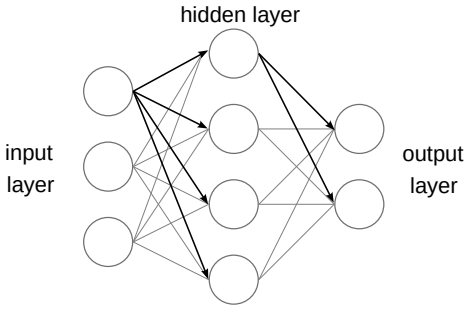


Figure 2.7: Schematic illustration of a simple multi-layer perceptron.

For our purposes, we can simply view an MLP as a function $f(\mathbf{x}; \Theta)$, where the parameter Θ contains all weights and bias terms which determine the shape of the mapping. With an increasing number of layers and neurons, an MLP can approximate arbitrarily complex functions.

The training of an MLP is carried out by minimizing the mean squared output error (cf. Eq. 1.9)

$$E = \frac{1}{N} \sum_i \|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i; \Theta)\|^2 \tag{2.42}$$

with respect to the parameters Θ. The involved gradients can be efficiently calculated by help of the backpropagation scheme (Rumelhart, Hinton, and Williams, 1986).

In an unsupervised setting, an MLP variant of particular interest is the so-called *auto-associative* network, or *auto-encoder*. Such a network has an equal number of input and output neurons and is trained with the objective that its output should be equal to the input. The most important ingredient of an auto-associative MLP is the *bottleneck layer*, that is, a hidden layer which has a smaller number of neurons than the input layer. If the auto-association (or input reconstruction) task has to be carried out through a bottleneck, the network somehow has to find a more compact representation of the data.

Auto-associative networks with three layers (input, bottleneck, and output) have been studied e.g. by Bourlard and Kamp (1988) and Baldi and Hornik (1989), who showed that a linear auto-encoder has a globally optimal solution which is equivalent to PCA.

Kramer (1991) derived a non-linear generalization of PCA by utilizing auto-associative MLPs with an input layer, an encoding layer with sigmoid activation, a linear bottleneck

layer, a decoding layer with sigmoid activation, and a linear output layer. An example of such a 5-layer network is depicted in Fig. 2.8. Deeper and non-symmetric auto-encoders are also possible: for example DeMers and Cottrell (1993) utilized a network with one hidden layer before, and two hidden layers behind the bottleneck layer.

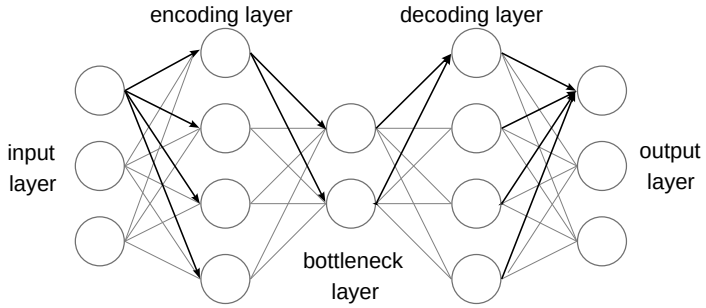


Figure 2.8: Schematic illustration of an auto-associative multi-layer perceptron.

If we denote the observed data vectors with $\mathbf{y}$ and the output of the bottleneck layer with $\mathbf{x}$, we can view the network as a composition of an encoding function $\mathbf{x} = \mathbf{g}(\mathbf{y})$, which maps the observable data to its compact representation, and a decoding function $\mathbf{y}' = \mathbf{f}(\mathbf{x})$, which maps back into the data space. The network is trained by minimizing the reconstruction error

$$E = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{y}_i - (\mathbf{f} \circ \mathbf{g})(\mathbf{y}_i; \Theta)\|^2 \tag{2.43}$$

with respect to the network parameters Θ. Through this, both mappings $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ are learned at the same time, and they can later be accessed by splitting up the network.

While the size of the bottleneck layer determines the dimensionality of the compact representations $\mathbf{x}_i$, the complexity of the mappings $\mathbf{f}(\cdot)$ and $\mathbf{g}(\cdot)$ is controlled by the number of hidden layers before and behind the bottleneck, and by the number of neurons in those layers. Kramer (1991) utilized the Akaike information criterion to select the size of the encoding and decoding layers. DeMers and Cottrell (1993) adjusted the size of the bottleneck by a pruning strategy, which is based on penalizing and measuring the variance of single bottleneck neurons.

The main drawback of auto-associative MLPs lies in the fact that networks of increasing depth and size (necessary for complex mappings) are increasingly hard to train and sensitive towards getting stuck in bad local minima of the objective function (2.43). Recently, however, Hinton and Salakhutdinov (2006) described how even deep auto-encoders can be created efficiently. They compose the network symmetrically from a series of restricted Boltzmann machines (Smolensky, 1986), which are pre-trained to reduce the dimensionality in a step-by-step manner. When endowing the auto-encoder with the pre-trained weights, the network is already close to a good minimum and can be fine-tuned by backpropagation.

2.4. The Self-Organizing Map

The Self-Organizing Map (Kohonen, 2001; Ritter, Martinetz, and Schulten, 1992) is a very popular unsupervised type of neural network. Since its introduction by Kohonen (1982), it has inspired a very large body of research, with applications in a broad range of disciplines. Kaski, Kangas, and Kohonen (1998) and M. Oja, Kaski, and Kohonen (2003) list 5384 related scientific papers published before the year 2003.

The standard Self-Organizing Map (SOM) consists of a set of K neurons that are organized on a fixed, usually two-dimensional grid. Each neuron $j = 1 \dots K$ is endowed with a reference or weight vector $\mathbf{w}_j \in \mathbb{R}^d$, which serves a similar purpose as in the vector quantization network (cf. Sec. 2.1.4): it acts as an approximation of nearby data vectors. Also the SOM learning rule is similar to the VQ rule (2.17), but differs in two important aspects. Firstly, if at any time t a data vector $\mathbf{y}^t$ is presented to the network, the neuron c with the closest weight $\mathbf{w}_c$ is determined, but nonetheless *all* weight vectors are updated. Secondly, the strength of the update depends on the location $\mathbf{r}_j$ of the neurons within the low-dimensional grid, that is, far away neurons (regarding the grid topology) get updated less strongly. In particular, the SOM rule is given by (Kohonen, 2001)

$$c = \arg\min_j \|\mathbf{y}^t - \mathbf{w}_j^t\| \tag{2.44}$$

$$\mathbf{w}_j^{t+1} = \mathbf{w}_j^t + h_{cj}^t \left(\mathbf{y}^t - \mathbf{w}_j^t\right) \ , \quad j = 1 \dots K \tag{2.45}$$

$$h_{cj}^t = \alpha(t) \exp\left(-\frac{1}{2\sigma^2(t)} \|\mathbf{r}_c - \mathbf{r}_j\|^2\right). \tag{2.46}$$

Here, both the "learning rate" $\alpha(t)$ and the "neighborhood range" $\sigma(t)$ are monotonously decreasing functions of time. The exponential in (2.46) is frequently chosen, but can be replaced by similarly shaped functions.

The desired effect of this learning rule is that nearby neurons (regarding the topology in the grid) obtain reference vectors that are close in data space. The other way around, if one introduces a mapping

$$\mathbf{g}(\mathbf{y}) = \mathbf{r}_c \quad \text{with} \quad c = \arg\min_j \|\mathbf{y} - \mathbf{w}_j\|, \tag{2.47}$$

which for a data vector $\mathbf{y}$ yields a discrete grid location, nearby data vectors should get mapped to nearby (or equal) grid positions. Thus, the SOM yields a topology preserving mapping between the data space and the discrete grid.

Figure 2.9 exemplarily shows a 16×14 SOM with 3-D reference vectors that was trained[3] to fit 2000 data points sampled from the surface of a hemisphere.

In contrast to PCA and auto-encoder networks, the SOM does not feature a well-defined objective function, which causes several problems. For example, one cannot easily compare the results of multiple training runs, or compare SOMs with different grid specifications. The learning rule is only heuristically defined, and some care has to be taken for adjusting the functions $\alpha(t)$ and $\sigma(t)$ during training. Whereas other

[3]The SOM was trained using the SOM toolbox for Matlab, which is available at http://www.cis.hut.fi/projects/somtoolbox/

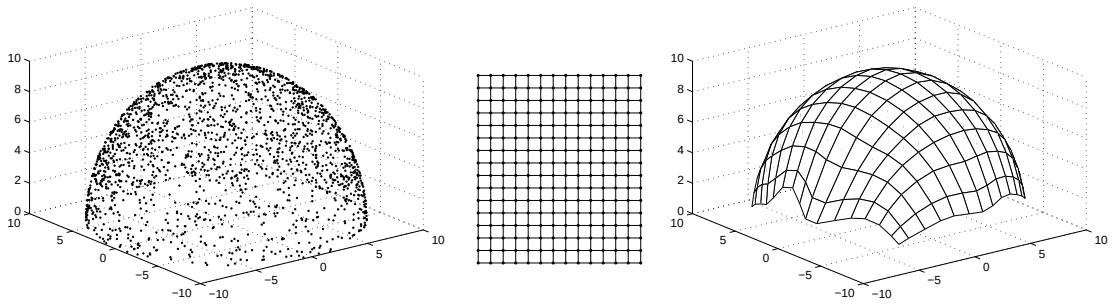


Figure 2.9: A simple SOM example. The left plot shows 2000 data points in 3-D that were sampled from the surface of a hemisphere. The middle plot depicts the SOM's rectangular grid with 16×14 nodes. The reference vectors of the SOM, together with their grid connections, are depicted in the right plot.

algorithms are guaranteed to converge to at least a local minimum of their objective function, training of the SOM is slowed and stopped by heuristically decreasing the learning rate. Furthermore, if the neighborhood range σ is too large, the SOM will be "stiff", because every update step influences a wide range of neurons in the same way. If σ is too small, the reference vectors of neighboring neurons may be torn apart, introducing topological errors. Kohonen (2001) also introduced a batch variant of the SOM, where the reference vectors are updated in a similar fashion as with the K-Means algorithm, and which thus does not need a learning rate parameter $\alpha(t)$. The neighborhood range or stiffness paramter $\sigma(t)$, however, still has to be adjusted properly.

2.4.1. Variants of the SOM

Koikkalainen and Oja (1990) introduced so-called "Self-Organizing Hierarchical Feature Maps", which can be coarsely described as a tree structure of multiple SOMs. The purpose of that model is to speed up the best-match search (2.44) while keeping the flexibility of a large number of nodes.

The Parametrized Self-Organizing Map (PSOM) is an extension of the SOM by Lagrangian interpolation of the reference vectors (Ritter, 1993). We will discuss the PSOM in detail in chapter 3, where also recent modifications to the algorithm will be presented.

Ritter (1999) also introduced a SOM model where the neurons form a lattice in the hyperbolic space $\mathbb{H}^2$, which shows some favorable properties as compared with lattices in the normal Euclidean 2-D space, and which can be favorably utilized for visualization purposes. A hierarchical hyperbolic SOM (H^2SOM) has been proposed by (Ontrup and Ritter, 2005).

A variant of the SOM which can handle structured data (SOM-SD) has been proposed by Hagenbuchner, Sperduti, and Tsoi (2003).

2.5. Principal curves

Principal curves have been introduced by Hastie (1984) and Hastie and Stuetzle (1989) as a nonlinear generalization of principal axes (PCA). Whereas a principal axis is a straight line that passes through the middle of a dataset (or distribution), a principal

curve is the range of a smooth nonlinear function $\mathbf{f} : I \subset \mathbb{R} \to \mathbb{R}^d$ with the same property. In the following, we assume that the curve is parametrized by its arc-length, which implies $\|\mathbf{f}'(x)\| = 1$ at any point x in a closed interval $I = [x_s; x_e] \subset \mathbb{R}$.

An important ingredient of the authors' definition is *self-consistency*, which says that any point $\mathbf{f}(x)$ of the principal curve must be the average of the data that are projected onto it. The projection of a data vector $\mathbf{y}$ onto the curve $\mathbf{f}(\cdot)$ is defined to yield the *projection index* (Hastie and Stuetzle, 1989)

$$x_{\mathbf{f}}(\mathbf{y}) = \sup_x \left\{ x : \|\mathbf{y} - \mathbf{f}(x)\| = \inf_{x'} \|\mathbf{y} - \mathbf{f}(x')\| \right\}, \tag{2.48}$$

which takes into account cases where multiple values for x yield the same projection distance $\|\mathbf{y} - \mathbf{f}(x)\|$. Treating $\mathbf{y}$ as a random vector, self-consistency can then be stated as

$$\mathbf{f}(x) = \langle \mathbf{y} \mid x_{\mathbf{f}}(\mathbf{y}) = x \rangle_{\mathbf{y}} \quad \forall x. \tag{2.49}$$

Hastie and Stuetzle (1989) show that under this definition, a principal curve is a critical point of the expected squared projection distance

$$D(\mathbf{f}) = \left\langle \inf_x \|\mathbf{y} - \mathbf{f}(x)\|^2 \right\rangle_{\mathbf{y}}, \tag{2.50}$$

that is $\frac{\partial}{\partial \lambda} D(\mathbf{f} + \lambda \Delta \mathbf{f})\big|_{\lambda=0} = 0$. This definition generalizes PCA, but has some difficulties. For example, *all* principal axes (also those corresponding to smaller PCA eigenvalues) are principal curves. Furthermore, if the data stems from a model $\mathbf{y} = \hat{\mathbf{f}}(x) + \mathbf{u}$ with zero-mean noise $\mathbf{u}$, in general the function $\hat{\mathbf{f}}(\cdot)$ itself is *not* a principal curve (Hastie and Stuetzle, 1989).

In order to fit a principal curve to a finite dataset $\{\mathbf{y}_i \mid i = 1 \dots N\}$, Hastie and Stuetzle (1989) simplified the model to a polygon with vertices $\mathbf{f}(x_1), \mathbf{f}(x_2) \dots \mathbf{f}(x_N)$, and proposed an iterative scheme based on the following alternating steps:

a) Project the data $\{\mathbf{y}_i\}$ onto the (fixed) polygon, determine new projection indices $\{x_i\}$, and re-parametrize to arc-length.

b) Adapt the vertices while keeping $\{x_i\}$ fixed, i.e. set $\mathbf{f}(x_i)$ to a locally weighted average $\sum_j w_{ij} \mathbf{y}_j$, where the weights w_{ij} are determined by a decreasing function of $|x_j - x_i|$.

Step b) corresponds to estimating the conditional expectation (2.49) from the data, where calculating a locally weighted average prevents the algorithm from resulting in the "interpolation solution" $\mathbf{f}(x_i) = \mathbf{y}_i$, and enforces the smoothness of the discretized curve. Please note, however, that the fitting criterion of this step only regards the distances between data vectors and vertices, and not those between data vectors and polygon segments.

The balance between closely fitting the data vectors and smoothness is adjusted by the span of the local averaging. Hastie and Stuetzle (1989) proposed to decrease the span in successive iterations and to finally select it by utilizing cross-validation. As an initial configuration, the vertices $\mathbf{f}(x_i)$ are set to the projections of the data vectors $\mathbf{y}_i$ onto the first principal axis as given by PCA. The algorithm is run until the relative

change of the mean projection distance is below a certain threshold. Hastie and Stuetzle (1989) did not provide a prove of convergence.

The discretized principal curve model is quite similar to a 1-D SOM, as has been investigated by Ritter et al. (1992). In particular, the vertex fitting step (b) resembles the vector quantization involved in the SOM.

2.5.1. Generative model

As an alternative model, Tibshirani (1992) defined the principal curve of a distribution $p_y(\mathbf{y})$ to be a triplet $(p_x(x), p_c(\mathbf{y}|x), \mathbf{f})$ with the three properties

$$p_y(\mathbf{y}) = \int p_c(\mathbf{y}|x)p_x(x)dx \quad (\text{“consistency”}) \tag{2.51}$$

$$p_c(\mathbf{y}|x) = \prod_{i=1}^{d} p_c(y_i|x) \quad (\text{“conditional independence”}) \tag{2.52}$$

$$\mathbf{f}(x) = \langle \mathbf{y}|x \rangle = \int \mathbf{y} p_c(\mathbf{y}|x)d^d y. \tag{2.53}$$

With this model, the data $\mathbf{y}$ are viewed to be generated from first drawing a latent variable x from a distribution $p_x(\cdot)$, and then drawing $\mathbf{y}$ from a conditional distribution $p_c(\mathbf{y}|x)$, the mean of which is a point $\mathbf{f}(x)$ on the curve. Now, by definition, the “generator” curve $\hat{\mathbf{f}}(x)$ mentioned in the last section *is* a principal curve.

For finding a principal curve of a particular dataset $\{\mathbf{y}_i\}$, one first has to specify a parametric model of the conditional density $p_c(\mathbf{y}|x)$, for example a Gaussian distribution with mean $\mathbf{f}(x)$ and a diagonal[4] covariance matrix $\mathbf{C}(x)$. Then, fitting of the model can be achieved by maximizing the log-likelihood

$$L(\Theta) = \log \prod_{i=1}^{N} p(\mathbf{y}_i; \Theta) = \sum_{i=1}^{N} \log \int p_c(\mathbf{y}_i|x; \Theta)p_x(x)dx. \tag{2.54}$$

The integral in (2.54) can be replaced by a sum, where the integrand is evaluated at N sample points $a_j,\ j = 1 \ldots N$ (Tibshirani, 1992; Lindsay, 1983). Correspondingly, also the parameters $\Theta = (\mathbf{f}(x), \mathbf{C}(x))$ only have to be estimated at these sample points, and will thus be denoted as $\Theta_j = \Theta(a_j)$ in the following. The resulting model is a mixture of Gaussians, with the number of components being equal to the number of data vectors.

Similarly to the case of clustering (Sec. 2.1.2), Tibshirani (1992) extended the dataset by a set of unobserved variables $\{x_i \mid i = 1 \ldots N\}$, where $x_i \in \{a_1, a_2, \ldots, a_N\}$ indicates by which component $\mathbf{y}_i$ was generated. He then proposed to maximize the log-likelihood of the complete data (cf. eq. 2.10)

$$L^{comp}(\Theta) = \log \prod_{i=1}^{N} p(\mathbf{y}_i, \mathbf{x}_i; \Theta_i) = \sum_{i=1}^{N} \log \Big(p_c(\mathbf{y}_i|x_i; \Theta_i)p_x(x_i) \Big) \tag{2.55}$$

via the EM algorithm (Dempster et al., 1977). In contrast to the clustering algorithm of section 2.1.2, this model has as many components as data vectors, which can lead to

[4] A diagonal covariance matrix corresponds to the assumption of conditional indepence.

severe overfitting. In particular, by setting the mean $\mathbf{f}(a_i)$ of the i-th mixture component to $\mathbf{y}_i$, the likelihood can be arbitrarily increased by decreasing the variance of the Gaussians. In the limit, this leads to a principal "curve" which only consists of the data vectors itself.

Tibshirani (1992) avoided this problem by the inclusion of a penalty term that enforces the smoothness of the curve. More precisely, he proposed to maximize the penalized log-likelihood

$$L_{pen}(\Theta) = L(\Theta) - (x_e - x_s) \sum_{i=1}^{d} \lambda_i \int_{x_s}^{x_e} \left(f_i''(x)\right)^2 dx, \tag{2.56}$$

where the factors λ_i balance between smoothness and closeness of fit. The maxima of (2.56) are calculated by applying cubic spline smoothing (Green and Silverman, 1994) to the mixture centers $\mathbf{f}(a_i)$ as determined without regularization. The locations a_i are optimized by a Newton-Raphson procedure, or rather by single Newton-Raphson steps within iterations of the EM algorithm (Tibshirani, 1992).

In comparison to the model of Hastie and Stuetzle (1989), the generative model has theoretical advantages, i.e. it overcomes the conceptual problem with the "generator" curve, but it has not been reported to perform better in practice (Tibshirani, 1992).

2.5.2. Polygonal lines

Kégl, Krzyzak, Linder, and Zeger (2000) proposed another formulation of principal curves and a corresponding algorithm. By their definition, a curve $\mathbf{f} : I \to \mathbb{R}^d$ is a *principal curve of length* L for a distribution $p(\mathbf{y})$, if it minimizes the expected squared distance (2.50) over all curves of length less or equal to L (Kégl et al., 2000). The authors proved that such a curve (which is not necessarily unique) exists for any $L > 0$, provided the distribution of $\mathbf{y}$ has finite second moments.

In order to derive an algorithm for fitting principal curves to datasets, Kégl et al. (2000) restricted the model of $\mathbf{f}$ to polygonal lines with K segments, where in contrast to the model of Hastie and Stuetzle (1989), the segments (and not only the vertices) are involved in the fitting process, and $K < N$. With such a restricted function class (denoted by $\mathbf{f}_K$), and under the assumption that the data are distributed only within a bounded convex set $B \subset \mathbb{R}^d$, Kégl et al. (2000) were able to prove that the empirical projection error (or empirical squared projection distance)

$$E_N(\mathbf{f}_K) = \frac{1}{N} \sum_{i=1}^{N} \min_x \|\mathbf{y}_i - \mathbf{f}_K(x)\|^2 \tag{2.57}$$

converges (for $N \to \infty$) against the expected projection error $D(\mathbf{f})$ (2.50) at a rate $O(N^{-\frac{1}{3}})$, if K is chosen to be proportional to $N^{\frac{1}{3}}$.

Despite the strong theoretical foundation, Kégl et al. (2000) could not prove the convergence of the actual algorithm they presented. Briefly summarized, the polygonal line algorithm starts with an optimal single line segment ($K = 1$), which coincides with the first principal axis (PCA). The length of this initial segment is adjusted such that it can hold the orthogonal projections of all data vectors $\mathbf{y}_i$.

The algorithm then runs in two nested loops. In the outer loop, the current configuration with K segments is optimized, after which K is compared to some heuristically defined threshold $c(N, E_N)$. If $K \geq c(N, E_N)$, the algorithm is terminated. Otherwise, a new vertex is added at the midpoint of the segment with the most data projections, and the next iteration is executed with an incremented value of K.

The optimization of the configuration for a fixed K takes place in an inner loop, where only one vertex $\mathbf{v}_j$ gets updated per iteration. The new location of that vertex is obtained by minimizing a weighted sum of the projection error and a penalty term, the latter being based on the length of the curve and the angle between the segments at $\mathbf{v}_j$. As a pre-factor of the penalty term, Kégl et al. (2000) used an experimentally determined constant.

Because of the reduced number of vertices, the computational complexity of the polygonal line algorithm is slightly lower than that of the original principal curve algorithm by Hastie and Stuetzle (1989). The practical curve fitting performance was reported to be "either comparable or better" (Kégl et al., 2000).

2.5.3. K-segments and local PCA

All aforementioned principal curves algorithms share a weakness, namely the dependance of the final curve model on its initialization. If the structure of the data is heavily twisted, using the principal axis as an initial configuration can lead to being trapped in a bad local minimum. As an example consider the spiral dataset depicted in Fig. 2.10. Such a dataset has been investigated by Kégl et al. (2000), who noted that the original principal curve algorithm of Hastie and Stuetzle (1989) always fails to detect the underlying structure, whereas their polygonal line algorithm succeeds in some cases, but only if the spiral is not too long (or rather, has too many windings).

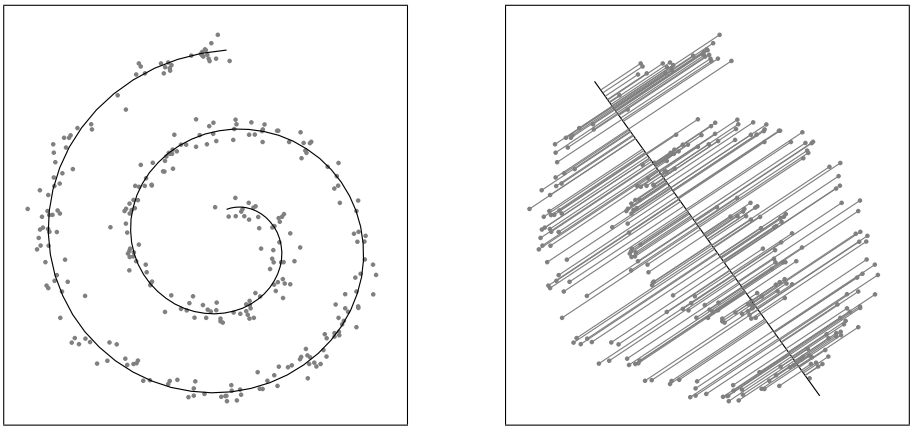


Figure 2.10: Illustration of a bad initialization as given by PCA. Left: underlying spiral (black curve) and sampled data points (gray dots). Right: projections (along gray lines) of the data onto the first principal axis (black line). In this case, PCA yields very bad projection indices: far away data points get projected onto the same regions of the principal axis, and thus a following optimization (e.g. by local smoothing) of the vertices will hardly improve the model.

To overcome this problem, Verbeek, Vlassis, and Kröse (2002, 2001) introduced an algorithm which can be roughly described as a combination of local PCA and polygonal

lines. The method works by alternating the steps of a) fitting unconnected segments (local principal axes), and b) connecting the segments to form a polygonal line. Depending on a performance criterion, a new local linear model is added before going back to step a).

Within step a), Verbeek et al. (2001) use an EM-like algorithm to optimize a mixture model, where each component is given by a line segment with a certain length $2a$. The posterior distribution $p(\mathbf{y}|j)$ of the data $\mathbf{y}$ given the j-th mixture component is a product of two factors. The first factor accounts for the projection distance onto the line, and is modelled as a 1-D Gaussian with variance σ^2. The second factor is a density function which decays like the same Gaussian if the projection onto the line yields a 1-D coordinate outside a range $[-a+\sigma; a-\sigma]$, but which is constant within that range. The product density thus has the form of a line segment, which is smeared in all directions of the data space. The update of the local models is similar to local PCA, where the new length parameter a is heuristically set to $3\sqrt{\lambda_1}$, where λ_1 is the largest eigenvalue of the local PCA model. The parameter σ is a global smoothing parameter, which has to be set by the user.

Within step b), Verbeek et al. (2001) seek a connection of the segments that minimizes the total length of the polygonal line, plus a term which penalizes sharp angles between adjacent segments, where again the user has to specify the corresponding pre-factor. Since there are $2^{K-1}K!$ possible ways to connect K segments, the authors propose to start with a greedy search algorithm, the result of which can optionally be refined later.

Figure 2.11 shows some K-segment principal curves that were fitted[5] to a spiral dataset. For moderate noise levels and a suitable maximal number K of segments, the algorithm is able to detect the underlying structure of the data. Local minima, however, are still a problem.

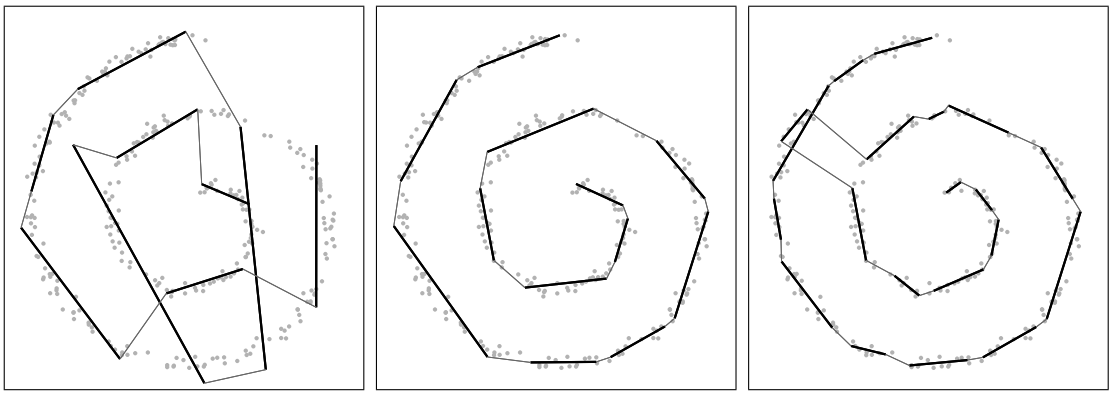


Figure 2.11: K-segments principal curves fitted to a spiral dataset (gray dots). The thick black lines indicate the local PCA models, whereas the thin gray lines depict the connections as determined by a heuristic search. From left to right: $K = 9$, $K = 12$, and $K = 20$.

[5]The K-segment curves in Fig. 2.11 were calculated using the MATLAB code written by J. J. Verbeek, which is available at http://carol.wins.uva.nl/∼jverbeek/ (Mar. 2007).

2.6. Principal surfaces

While the last section was devoted to algorithms that can find one-dimensional manifolds, the focus of this section is on algorithms for higher-dimensional structures. Thereby, the cases $q = 2$ and sometimes $q = 3$ are of particular practical interest, because such a model of the data can be favorably used for visualization purposes, for example in exploratory data analysis (see e.g. Ripley, 2005).

As a first definition of principal surfaces, Hastie (1984) and Hastie and Stuetzle (1989) generalized their concept of principal curves. In particular, their formulas for projecting the data, self-consistency and total projection distance (2.48–2.50) carry over to the 2-D case if the projection index $x \in \mathbb{R}$ is replaced by $\mathbf{x} \in \mathbb{R}^2$.

However, not every idea that works in 1-D can be applied in 2-D, too. For example, the projection onto line segments of a curve does not have a simple analogue for a surface, and local smoothing for adapting the vertices is more complex, too (Hastie, 1984).

Furthermore, while the arc-length of a principal curve can be identified with the area of a principal surface, the latter is much harder to handle (e.g. in a "maximum area" constraint), and a re-parametrization of $\mathbf{x}$ to "unit speed" $\|\mathbf{f}'(\mathbf{x})\| = 1$ is impossible in general. Thus, principal curve methods which explicitly exploit geometric properties like the polygonal lines and the K-segments algorithm cannot be generalized easily.

2.6.1. The Generative Topographic Mapping

The Generative Topographic Mapping (GTM) was proposed as a possible SOM replacement by Bishop, Svensén, and Williams (1997, 1998a, 1998b). The central aspect of the GTM is the assumption that the observed data $\{\mathbf{y}_i\}$ are "generated" from a parametric model

$$\mathbf{y} = \mathbf{f}(\mathbf{x}; \mathbf{W}) + \mathbf{u} \ , \quad \mathbf{u} \sim \mathcal{N}(0, \beta^{-1}\mathbf{I}_d), \tag{2.58}$$

where $\mathbf{x} \in \mathbb{R}^q$ is a latent variable, $\mathbf{W}$ is a parameter matrix, and $\mathbf{u}$ denotes Gaussian noise with variance β^{-1}. Note that if $\mathbf{f}(\cdot)$ is restricted to be linear, and the distribution of $\mathbf{x}$ is Gaussian, then this model is identical to probabilistic PCA (Sec. 2.2.1).

In order to derive an algorithm which is "similar in spirit to the SOM", Bishop et al. (1998b) chose a generalized linear regression model

$$\mathbf{f}(\mathbf{x}) = \mathbf{W}\mathbf{b}(\mathbf{x}), \tag{2.59}$$

where $\mathbf{b}(\cdot) \in \mathbb{R}^M$ is a vector of *fixed* basis functions, $\mathbf{W} \in \mathbb{R}^{d \times M}$ is a parameter matrix, and the distribution of the latent variables has the special form

$$p(\mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} \delta(\mathbf{x} - \mathbf{r}_k). \tag{2.60}$$

Here, the vectors $\mathbf{r}_k$ denote coordinates of K nodes of a regular grid, directly corresponding to the SOM formulation (cf. Sec. 2.4). For their experiments, Bishop et al. (1998b) chose the basis functions to be radially symmetric Gaussians with a common width parameter σ, whose centers where evenly distributed across the grid. Note that the number M and locations of these centers does not have to be equal to the set $\{\mathbf{r}_k\}$.

Given this model, the log-likelihood of observing a dataset $\{\mathbf{y}_i \mid i = 1 \dots N\}$ can be written as

$$\begin{aligned} L(\mathbf{W}, \beta) &= \log \prod_{i=1}^{N} p(\mathbf{y}_i|\mathbf{W}, \beta) \\ &= \sum_{i=1}^{N} \log \int p(\mathbf{y}_i|\mathbf{x}, \mathbf{W}, \beta) p(\mathbf{x}) d\mathbf{x} \\ &= \sum_{i=1}^{N} \log \left[\frac{1}{K} \sum_{j=1}^{K} p(\mathbf{y}_i|\mathbf{r}_j, \mathbf{W}, \beta) \right] \\ &= \sum_{i=1}^{N} \log \left[\frac{1}{K} \sum_{j=1}^{K} \left(\frac{\beta}{2\pi} \right)^{\frac{d}{2}} \exp \left(-\frac{\beta}{2} \|\mathbf{y}_i - \mathbf{W}\mathbf{b}(\mathbf{r}_j)\|^2 \right) \right]. \end{aligned} \tag{2.61}$$

The model of the GTM can thus be viewed as a constrained mixture of Gaussians that are centered at $\mathbf{W}\mathbf{b}(\mathbf{r}_j)$. Similar to the SOM, which can be viewed as a constrained vector quantizer, the constraint lies in the topological relation of the nodes $\mathbf{r}_j$ within the grid. Thus, as a slogan, the GTM is to the SOM what the EM clustering algorithm is to Kohonen's VQ rule.

The GTM model is fitted by maximizing the likelihood via the EM algorithm, where the E-step consists of calculating the posterior probabilities of the j-th mixture component having generated the data $\mathbf{y}_i$ given the current parameters. These "responsibilities" are given by (Bishop et al., 1998b)

$$R_{ji}^{old} = P(\mathbf{r}_j|\mathbf{y}_i, \mathbf{W}, \beta) = \frac{p(\mathbf{y}_i|\mathbf{r}_j, \mathbf{W}, \beta)}{\sum_{k=1}^{K} p(\mathbf{y}_i|\mathbf{r}_k, \mathbf{W}, \beta)}. \tag{2.62}$$

The M-step consists of maximizing the expected log-likelihood of the complete data (including the "generation" responsibilites)

$$\langle L^{comp}(\mathbf{W}, \beta) \rangle = \sum_{i=1}^{N} \sum_{j=1}^{K} R_{ji}^{old} \log p(\mathbf{y}_i|\mathbf{r}_j, \mathbf{W}, \beta) \tag{2.63}$$

with respect to the parameters, which yields (cf. Bishop et al., 1998b)

$$\mathbf{B}\mathbf{G}\mathbf{B}^T\mathbf{W}_{new}^T = \mathbf{B}\mathbf{R}^{old}\mathbf{Y}^T \tag{2.64}$$

$$\frac{1}{\beta^{new}} = \frac{1}{dN} \sum_{i=1}^{N} \sum_{j=1}^{K} R_{ji}^{old} \|\mathbf{y}_i - \mathbf{W}_{new}\mathbf{b}(\mathbf{r}_j)\|^2. \tag{2.65}$$

Here, $\mathbf{Y}$ denotes the $d \times N$ data matrix $(\mathbf{y}_1, \mathbf{y}_2 \dots \mathbf{y}_N)$, the $K \times N$ matrix $\mathbf{R}^{old}$ holds the components R_{ji}^{old}, the $M \times K$ matrix of basis functions $\mathbf{B}$ has components $(\mathbf{B})_{mk} = b_m(\mathbf{r}_k)$, and $\mathbf{G}$ is a diagonal matrix with K entries $g_{kk} = \sum_{i=1}^{N} R_{ki}^{old}$.

For initializing the model, Bishop et al. (1998b) adjust the weight matrix $\mathbf{W}$ such that the GTM manifold (the range of $\mathbf{f}(\mathbf{x}; \mathbf{W})$) equals the manifold of a PCA solution. The

inverse noise variance β^{-1} is initially set to the maximum of λ_{q+1} and $(\frac{D}{2})^2$, where λ_{q+1} denotes the largest "uncaptured" PCA eigenvalue, and D denotes the distance between grid points, or rather their images $\mathbf{f}(\mathbf{r}_i; \mathbf{W})$ within the linear manifold. In contrast to the SOM, the GTM algorithm is guaranteed to converge to a local minimum, and does not need a specification of a time-varying neighborhood range. Multiple learning runs can be compared with respect to their results for the log-likelihood.

When a data vector $\mathbf{y}$ is projected onto the manifold, the SOM just returns the grid coordinate of the discrete best match search (2.44). Within the GTM, the lower dimensional representation $\mathbf{x}$ is governed by the posterior $p(\mathbf{x}|\mathbf{y})$. The mode of this posterior corresponds to the discrete best match, but a more natural representation is given by the mean, that is, the conditional expectation $\langle \mathbf{x} \mid \mathbf{y} \rangle$.

Bishop et al. (1998a) describe how an additional prior $p(\mathbf{W}) = \mathcal{N}(0, \alpha\mathbf{I})$ on the space of weights can be incorporated, and also how Bayesian inference can be utilized to select suitable values for the parameters σ (width of the basis functions), α (prior distribution of weights) and β (inverse noise variance), in case the latter should not be estimated within the EM iterations (2.65). The authors also compare the GTM to a similar probabilistic variant of the SOM, which was proposed by Utsugi (1997a, 1997b).

2.6.2. Regularized Principal Manifolds

Smola, Mika, Schölkopf, and Williamson (2001) presented a framework for fitting principal curves and surfaces with strong theoretic foundations in statistical learning theory and functional analysis. In the following, their framework will be briefly summarized.

Similar to Hastie and Stuetzle (1989), the authors state the task of manifold learning as the minimization of the expected quantization (or reconstruction) error functional

$$E_{exp}(\mathbf{f}) = \left\langle \min_{\mathbf{x}} L(\mathbf{y}, \mathbf{f}(\mathbf{x})) \right\rangle_{\mathbf{y}} \tag{2.66}$$

where $L(\cdot, \cdot)$ is a loss function, e.g. the squared Euclidean error $\|\mathbf{y} - \mathbf{f}(\mathbf{x})\|^2$. Like in supervised learning (Sec. 1.3.2), estimating $\mathbf{f}(\cdot)$ from a finite dataset requires restricting the function class and some means of regularization, because otherwise $\mathbf{f}(\cdot)$ might overfit the training set.

Thus, the authors propose to minimize the regularized quantization functional

$$E_{reg}(\mathbf{f}) = \frac{1}{N} \sum_{i=1}^{N} \min_{\mathbf{x}_i} L(\mathbf{y}_i, \mathbf{f}(\mathbf{x}_i)) + \lambda Q(\mathbf{f}), \tag{2.67}$$

where the first summand is the empirical quantization error, and $Q(\mathbf{f})$ is a convex non-negative regularization term. A possible choice for $Q(\cdot)$ is the quadratic functional $Q(\mathbf{f}) = \frac{1}{2}\|\mathcal{P}\mathbf{f}\|^2_{\mathcal{H}}$, where $\mathcal{P}$ is a regularization operator which maps $\mathbf{f}$ into an Hilbert space $\mathcal{H}$. As an example, consider $\mathcal{P} = \frac{\partial}{\partial x}$ and $\mathcal{H}$ to be the space of square-integrable functions $\mathbf{f} : I \subset \mathbb{R} \rightarrow \mathbb{R}^d$, yielding

$$Q(\mathbf{f}) = \frac{1}{2}\|\mathcal{P}\mathbf{f}\|^2 = \int_I \|\mathbf{f}'(x)\|^2 dx, \tag{2.68}$$

which can be identified with the squared path length of a curve $\mathbf{f}$ (Smola et al., 2001). In this way, the framework subsumes the principal curves of finite length (cf. Sec. 2.5.2).

Relating to the connection between Mercer kernels (cf. Sec. 1.5.4) and regularization operators (Smola, Schölkopf, and Müller, 1998a), the authors propose to restrict $\mathbf{f}(\cdot)$ to kernel expansions of the form

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}_0 + \sum_{j=1}^{M} \mathbf{a}_j k(\mathbf{r}_j, \mathbf{x}) \tag{2.69}$$

with adaptable parameters $\mathbf{f}_0, \mathbf{a}_j \in \mathbb{R}^d$ and fixed pre-specified nodes $\mathbf{r}_j \in \mathbb{R}^q$. If the kernel $k(\cdot,\cdot)$ matches the regularization operator $\mathcal{P}$, then the regularization term can be expressed as

$$Q(\mathbf{f}) = \frac{1}{2} \sum_{i=1}^{M} \sum_{j=1}^{M} (\mathbf{a}_i^T \mathbf{a}_j) k(\mathbf{r}_i, \mathbf{r}_j), \tag{2.70}$$

where we assume that the constant function $\mathbf{f}_0$ is not penalized, that is $Q(\mathbf{f}_0) = 0$. The other way around, a specific kernel $k(\cdot,\cdot)$ induces a corresponding operator $\mathcal{P}$, for example the Gaussian kernel (1.32) penalizes all derivatives of the kernel expansion (2.69). As an important feature, note that the regularization term $Q(\mathbf{f})$ is a quadratic function of the expansion coefficients $\mathbf{a}_j$.

After choosing a Mercer kernel $k(\cdot,\cdot)$ and a set of nodes $\{\mathbf{r}_j\}$, a straightforward way to fit a Regularized Principal Manifold (RPM) is given by alternating the steps of

a) projecting the data vectors $\{\mathbf{y}_i\}$ onto the manifold, yielding $\{\mathbf{x}_i\}$, and

b) adapting the expansion coefficients $\{\mathbf{a}_j\}$ by minimizing (2.67) with fixed $\{\mathbf{x}_i\}$. In case the squared Euclidean loss is utilized, the optimal coefficients can be calculated by using linear algebra.

Comparing this algorithm to the GTM, one can identify many parallels despite the quite different backgrounds. Both algorithms require the choice of basis functions or a kernel and a set of sample points $\{\mathbf{r}_j\}$, and both algorithms decrease their objective function by two alternating steps. Here, the E-step of GTM can be viewed as a probabilistic projection, corresponding to step a), whereas the M-step of GTM matches the adaption step b).

2.6.3. Further methods

LeBlanc and Tibshirani (1994) proposed a principal surface model based on piecewise linear basis functions, and products thereof. Fitting of the model starts from a PCA solution and is then accomplished by alternating projection and adaption steps. If needed, the model complexity is increased by adding further basis function terms. The model is assessed by an objective function involving the squared projection error and the effective number of parameters.

K. Chang and Ghosh (2001) presented a unifying model for probabilistic principal surfaces, which revealed a connection between their own method (K. Chang and Ghosh, 1999), the GTM, and the manifold-aligned GTM (Bishop et al., 1998a) via a shape parameter of the noise model in (2.58).

2.7. Pointwise embedding methods

In the following two sections, we describe a family of algorithms for dimensionality reduction, which do not feature an explicit functional relationship between the observed data $\mathbf{y}_i \in \mathbb{R}^d$ and their lower-dimensional representations $\mathbf{x}_i \in \mathbb{R}^q$. Whenever the term "mapping" comes up, it will mean only a pointwise replacement of a data vector $\mathbf{y}_i$ by its representation $\mathbf{x}_i$.

This section briefly reviews classical and iterative approaches, which can all be viewed as instances of multi-dimensional scaling (MDS). The goal of MDS is to find a low-dimensional representation $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$ from a given symmetric $N \times N$ matrix $\mathbf{D} = (D_{ij})$ of distances or dissimilarities, such that the pairwise distances $d(\mathbf{x}_i, \mathbf{x}_j)$ are in good agreement with the given values D_{ij} (see e.g. Härdle and Simar, 2003).

2.7.1. Multi-dimensional scaling

Classical metric multi-dimensional scaling starts from a distance matrix $\mathbf{D}$ and seeks an Euclidean embedding with the property $D_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$. If such an embedding exists, that is, if $\mathbf{D}$ is a matrix of Euclidean distances, then the matrix $\mathbf{B}$ with elements

$$b_{ij} = -\frac{1}{2}\left(D_{ij}^2 - \frac{1}{N}\sum_k D_{kj}^2 - \frac{1}{N}\sum_k D_{ik}^2 + \frac{1}{N^2}\sum_{k,l} D_{kl}^2\right) \tag{2.71}$$

is positive semidefinite and can be expressed by dot products

$$b_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j \tag{2.72}$$

between centered vectors $\{\mathbf{x}_i\}$ (Härdle and Simar, 2003). These vectors can then be retrieved via an eigendecomposition of $\mathbf{B}$, which is equivalent to the dot product variant of PCA (cf. Sec. 2.2.3).

The number of positive eigenvalues determines the necessary dimensionality of a perfect embedding. Including fewer eigenvalues has the same effect as in PCA, that is, only the projections of the data onto a linear subspace are recovered, and components orthogonal to this subspace are lost.

If the distances $\mathbf{D}$ are non-euclidean, then $\mathbf{B}$ will have negative eigenvalues, which in classical MDS are just clipped to zero. The distances cannot be preserved in this case.

For large datasets, the eigendecomposition of the dot product matrix $\mathbf{B}$ can become impractical. To overcome this problem, de Silva and Tenenbaum (2004) introduced a variant of MDS where only a smaller set of "landmark" points are embedded via an eigendecomposition. The remaining data points are then placed by a triangulation procedure, using the distance information to the landmark points.

A more general form of metric MDS aims at minimizing an error measure (Quist and Yona, 2004)

$$E = \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} w_{ij}\left(D_{ij} - d(\mathbf{x}_i, \mathbf{x}_j)\right)^2, \tag{2.73}$$

where the terms w_{ij} can weight the influence of single distortions. Another popular choice is to use the distances in squared form. By including weights, the approach offers

more flexibility and can handle nonlinear structures in the data. However, in general the solution cannot be calculated by an eigendecomposition anymore, but rather iterative schemes have to be applied.

If the matrix $\mathbf{D}$ contains general dissimilarities, perserving only the rank of the pairwise dissimilarities might be preferable. This is the strategy of non-metric MDS algorithms, where the dissimilarities and distances can be transformed by any monotonous functions $f(\cdot)$ and $g(\cdot)$, which may even vary during training. The corresponding error measure is of the form (Quist and Yona, 2004)

$$E = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} w_{ij} \left(f(D_{ij}) - g(d(\mathbf{x}_i, \mathbf{x}_j)) \right)^2 . \tag{2.74}$$

In a manifold learning context, however, we assume that we are given real distances, so non-metric MDS will not be considered any further. An overview of MDS techniques including the historical development has been given by Shepard (1980). Recently, Quist and Yona (2004) have introduced a variant of MDS which mainly aims at preserving the cluster structure of the data. Their objective function is not based directly on distance distortions, but rather on the similarity between the distributions of distances.

2.7.2. The Sammon mapping

Sammon (1969) derived a famous non-linear variant of metric MDS, which is based on the distortion measure

$$E_{Sam} = \frac{1}{\sum_{i<j} D_{ij}} \sum_{i<j} \frac{\left(D_{ij} - \|\mathbf{x}_i - \mathbf{x}_j\| \right)^2}{D_{ij}}, \tag{2.75}$$

where D_{ij} denotes the distance between two data vectors $\mathbf{y}_i$ and $\mathbf{y}_j$, and the summation $\sum_{i<j}$ is to be understood in the sense of (2.73).

Since all error contributions $(D_{ij} - \|\mathbf{x}_i - \mathbf{x}_j\|)^2$ are down-weighted by the corresponding data space distance D_{ij}, the Sammon mapping primarily aims at preserving *small* distances. The underlying idea is that even when the global data distribution is highly nonlinear, small enough patches of it are almost linear, so locally the distances can be preserved. Figure 2.12 illustrates this reasoning.

Starting from randomly chosen initial coordinates $\{\mathbf{x}_i\}$, Sammon (1969) minimized (2.75) by gradient-descent or more precisely by a simplified Newton algorithm. The objective function (2.75) is non-convex, so the Sammon mapping suffers from the problem of local minima. The dimensionality of $\mathbf{x}$ has to be specified in advance.

2.7.3. Curvilinear Component Analysis

Curvilinear Component Analysis (CCA) was introduced by Demartines and Hérault (1997) as an improvement to the SOM. Indeed both algorithms share some concepts, but the core of CCA is much closer to metric MDS.

The first step of the CCA algorithm is to run vector quantization on a given dataset. This does not reduce the dimensionality of the data but the computational complexity

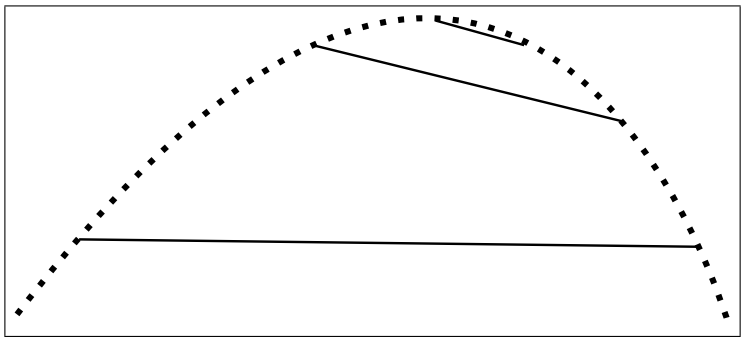


Figure 2.12: Illustration of the idea behind the Sammon mapping. The data (depicted by dots) cannot be embedded in 1-D without distorting their pairwise distances. By focussing on the preservation of small distances, the local structure of the data can be captured.

of further operations, which only process the code-book vectors[6]. In the following, N denotes the number of code-book vectors, and D_{ij} denotes the Euclidean distance between code-book vectors $\mathbf{c}_i$ and $\mathbf{c}_j$.

In the second step, CCA seeks to find low-dimensional coordinates $\{\mathbf{x}_i \mid i = 1 \dots N\}$ with distances $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|$, such that the following objective function is minimized (Demartines and Hérault, 1997):

$$E = \frac{1}{2} \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} (D_{ij} - d_{ij})^2 F(d_{ij}, \lambda). \tag{2.76}$$

This error measure has the form of general metric MDS (2.73). As compared to the Sammon mapping, the weights $w_{ij} = F(d_{ij}, \lambda)$ do not depend on the distances D_{ij} in the original space, but on d_{ij} in the low-dimensional space. $F(\cdot)$ plays a similar role to the factor h_{ji}^t of the SOM (2.46), and may be any bounded and monotonously decreasing function. Demartines and Hérault (1997) chose a simple step function

$$F(d_{ij}, \lambda) = \begin{cases} 1 & d_{ij} \leq \lambda \\ 0 & d_{ij} > \lambda, \end{cases} \tag{2.77}$$

by which only the preservation of distances up to a threshold λ is taken into account. Similar to the neighborhood range of the SOM, the parameter λ is crucial for finding a good mapping and has to be decreased during training. The schedule for varying λ is based on heuristics, and Demartines and Hérault (1997) even suggested that λ should be varied interactively by the user.

In order to minimize (2.76), Demartines and Hérault (1997) proposed an iterative scheme where in each step one vector $\mathbf{x}_i$ is randomly picked out, and the remaining vectors $\mathbf{x}_j,\ j \neq i$ are updated by

$$\mathbf{x}_j^{new} = \mathbf{x}_j^{old} - \alpha(t) \nabla_{\mathbf{x}_j} \left[(D_{ij} - d_{ij})^2 F(d_{ij}, \lambda) \right], \tag{2.78}$$

[6] In fact, such a pre-processing step has also been considered by Sammon (1969), who at that time noted that already 250 data points lead to a high computational burden.

where $\alpha(t)$ denotes a time-varying learning rate. On average, but not necessarily in every step, this update rule decreases the overall error (2.76). Demartines and Hérault (1997) argued that the stochastic nature results in finding better local minima, but they did not provide a proof of convergence.

CCA also provides a mechanism for mapping new data points $\mathbf{y}$ into the low-dimensional space. This is achieved by an iterative minimization of an error function similar to (2.76):

$$E(\mathbf{x}) = \sum_{j=1}^{N} (D_j - d_j)^2 F(d_j, \lambda), \tag{2.79}$$

where $D_j = \|\mathbf{y} - \mathbf{c}_j\|$ and $d_j = \|\mathbf{x} - \mathbf{x}_j\|$. By swapping the roles of $\mathbf{x}$ and $\mathbf{y}$, a corresponding reverse mapping back into data space can also be performed.

2.7.4. Curvilinear Distance Analysis

Lee, Lendasse, Donckers, and Verleysen (2000) proposed an improved variant of CCA, which mainly differs in the way of measuring distances in data space, and which thus was called Curvilinear Distance Analysis (CDA).

As the main improvement, after the vector quantization step, CDA builds a graph whose vertices are the code-book vectors. To this end, Lee et al. (2000) proposed to connect any two code-book vectors if they are the closest ones for a given original data vector – this is similar to the strategy used in the Neural-Gas algorithm. After that, the "curvilinear" distances $\tilde{D}_{ij}$ between code-book vectors $\mathbf{c}_i$ and $\mathbf{c}_j$ are calculated *along the graph*. In particular, if $\mathbf{c}_i$ and $\mathbf{c}_j$ are directly connected, $\tilde{D}_{ij}$ is set to $\|\mathbf{c}_i - \mathbf{c}_j\|$. Otherwise, $\tilde{D}_{ij}$ is the sum of the lengths of all connections in the shortest path between the vertices i and j. An illustration of the difference between the resulting curvilinear and the Euclidean distance is provided by Fig. 2.13.

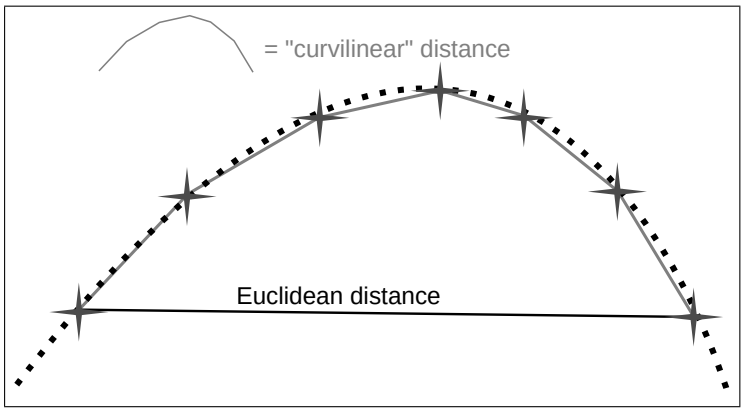


Figure 2.13: Illustration of the "curvilinear" distance measure of CDA. Code-book vectors are depicted by black stars, data vectors by black dots. While the Euclidean distance between the two code-book vectors at the bottom is given by the straight black line, CDA measures that distance by the total length of the connecting lines (drawn in gray).

In regions where the structure of the data is already linear (with a desired embedding space of at least 2-D), utilizing curvilinear distances $\tilde{D}_{ij}$ will not improve, but actually

worsen the mapping: in that case, the "right" measure for the distance between two vertices is given by a straight line, and not by a zigzag connection on the graph. To cope with this problem, Lee et al. (2000) introduced a dynamically adapted weighting factor $\omega(t)$ for balancing between the two distance measures, and set

$$D_{ij} = (1 - \omega(t)) \|\mathbf{c}_i - \mathbf{c}_j\| + \omega(t)\tilde{D}_{ij}. \tag{2.80}$$

As a further improvement over CCA, Lee et al. (2000) proposed some rules for automatically choosing the model parameters, such that a given "tolerable loss" is not exceeded. In particular, they propose an error threshold for the vector quantization step, the selection of the dimension q by local PCA, and formulas for the three time varying parameters α (learning rate), λ (neighborhood threshold) and ω (distance balancing). However, a sound theoretical foundation of these rules is not presented. In a later publication (Lee, Lendasse, and Verleysen, 2004), the balancing factor $\omega(t)$ is not utilized anymore, and the authors stress the robustness of the method towards the choice of learning rate and neighborhood threshold.

2.8. Nonlinear spectral embedding methods

This section describes a family of algorithms which can calculate pointwise embeddings $\{\mathbf{x}_i\}$ by an eigendecomposition of an $N \times N$ matrix. The exact definition of that matrix differs among the algorithms, but in any case, the first step is to build a neighborhood graph whose vertices are the observed data vectors $\{\mathbf{y}_i \mid i = 1 \dots N\}$, and whose edges are chosen such that each data point $\mathbf{y}_i$ is connected either

- to its K nearest neighbors, or
- to all data vectors that lie within a radius ϵ.

In both cases, the parameter K or ϵ has to be chosen large enough, so that the resulting neighborhood graph is connected, i.e. any vertex $\mathbf{y}_i$ must be reachable from any other vertex $\mathbf{y}_j$ via a path on the graph.

2.8.1. Isomap

The Isomap algorithm (Tenenbaum, Silva, and Langford, 2000) combines classical MDS and the measurement of distances along a neighborhood graph. Through this, it can cope with nonlinearly structured data, and the optimal embedding is given by the solution of an eigendecomposition.

The algorithm consists of three steps, the first of which is the generation of the aforementioned neighborhood graph. In the second step, similar to the CDA algorithm (Sec. 2.7.4), the $N \times N$ distance matrix $\mathbf{D}$ is computed on the neighborhood graph, i.e. D_{ij} is set to the length of the shortest path between data vectors $\mathbf{y}_i$ and $\mathbf{y}_j$. If the graph is not connected (e.g. from a too small choice of K), the distances between data vectors belonging to different isolated patches are undefined, and Isomap cannot embed all data points in one go. An efficient algorithm for computing the shortest path lengths has been introduced by Dijkstra (1959).

The third step of Isomap is to apply classical MDS on the matrix $\mathbf{D}$, which now contains not Euclidean distances, but rather curvilinear or "geodesic" distances. As in PCA and classical MDS, the necessary dimensionality of the embedding can be determined from the eigenvalue spectrum. For large datasets, the landmark trick of classical MDS can also be applied to Isomap (de Silva and Tenenbaum, 2004).

The mathematical formulation of Isomap is very elegant, and it can be proved that asymptotically (in the limit of infinite data), Isomap is guaranteed to recover the true manifold structure if its "intrinsic geometry is that of a convex region of Euclidean space" (Tenenbaum et al., 2000). The basic idea of that proof is that with increasing sampling density, the graph distances get closer to the true geodesic distances along the underlying manifold (cf. Fig. 2.13 and 2.14). If that manifold is not convex, the paths on the graph detour, so the estimated distances are too large[7].

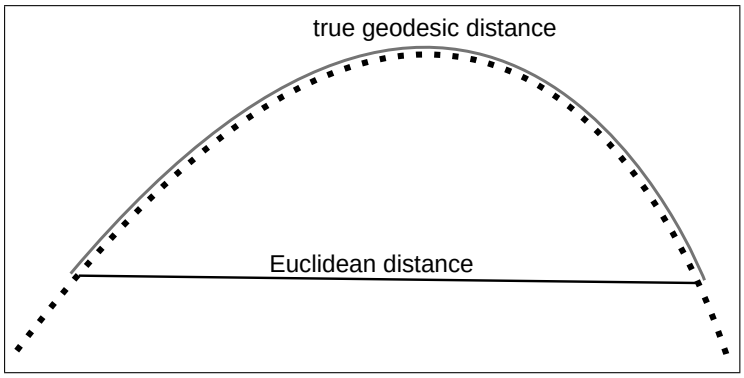


Figure 2.14: If the manifold is densely sampled (black dots), the length of paths on the graph yield an increasingly good approximation of the true geodesic distances (gray curve).

2.8.2. Locally Linear Embedding

Locally Linear Embedding (LLE), which has been introduced by Roweis and Saul (2000), is directly based on the observation that the structure of any manifold is approximately linear, provided one only looks at a small enough patch. Within such a patch, any data vector $\mathbf{y}_i$ can be reconstructed by a linear combination of its neighbors. A lower dimensional representation should have the same local structure, and thus a vector $\mathbf{x}_i$ should be reconstructible from the same linear combination of the corresponding points in the low dimensional space.

LLE implements this idea in three stages. After building a neighborhood graph as described at the beginning of section 2.8, the reconstruction error

$$E(\mathbf{W}) = \sum_{i=1}^{N} \left\| \mathbf{y}_i - \sum_{j=1}^{N} w_{ij} \mathbf{y}_j \right\|^2 \tag{2.81}$$

[7] As an illustration, imagine a sheet of paper with a hole in it. When rolled up a bit, the sheet is a 2-D manifold embedded in the 3-D space. The geodesic distances between two points on opposite sides of the hole will appear larger than they would be without the hole, since any path between the two points detours around it.

is minimized with respect to the weights w_{ij}, subject to the constraints that $w_{ij} = 0$ if $\mathbf{y}_j$ is not a neighbor of $\mathbf{y}_i$, and that $\sum_j w_{ij} = 1$ for all i. The resulting weights are invariant under rotation, scaling and translation of the data vectors, and thus capture the "intrinsic geometric properties of each neighborhood" (Roweis and Saul, 2000).

In the third stage of the algorithm, a corresponding reconstruction error

$$E(\mathbf{X}) = \sum_{i=1}^{N} \left\| \mathbf{x}_i - \sum_{j=1}^{N} w_{ij} \mathbf{x}_j \right\|^2 = \left\| \mathbf{X}^T - \mathbf{W}\mathbf{X}^T \right\|_F^2 = \operatorname{tr}\left(\mathbf{X}\mathbf{M}\mathbf{X}^T\right) \tag{2.82}$$

is minimized with respect to the low dimensional coordinates $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$, where $\mathbf{M} = (\mathbf{I}_N - \mathbf{W})^T(\mathbf{I}_N - \mathbf{W})$. To account for the invariance of the weights, we demand that the low dimensional coordinates are centered, uncorrelated and have unit variance. Under these constraints, the unique minimum of (2.82) is reached if the rows of $\mathbf{X}$ are the eigenvectors that belong to the eigenvalues $\lambda_2, \lambda_3, \ldots, \lambda_{q+1}$ of the matrix $\mathbf{M}$. In contrast to earlier algorithms, here the eigenvectors are assumed in increasing order, that is, λ_2 denotes the second *smallest* eigenvalue. The smallest eigenvalue λ_1 belongs to an eigenvector which violates the centering constraint.

In a later publication, Saul and Roweis (2003) point out that the magnitude of the eigenvalues cannot be utilized to reliably select the dimensionality of the embedding. Recently, H. Chang and Yeung (2006) introduced a variant of LLE which is robust against outliers in the data.

Another important difference to earlier algorithms is that the matrix $\mathbf{M} \in \mathbb{R}^{N \times N}$ is sparse, i.e. most of its elements are zero. This is a direct consequence of the sparseness of $\mathbf{W} \in \mathbb{R}^{N \times N}$, which has at most K non-zero elements in every row, provided the adjacency graph was built from a K nearest neighbors selection. If supported by the utilized linear algebra routines, sparseness of a matrix can dramatically enhance the computational and storage efficiency (see e.g. Gilbert, Moler, and Schreiber, 1992).

2.8.3. Maximum Variance Unfolding

In the Semidefinite Embedding (SDE) algorithm (Weinberger, Sha, and Saul, 2004; Weinberger and Saul, 2006), later called Maximum Variance Unfolding (MVU), the lower dimensional coordinates are retrieved as the top eigenvectors of a dot product matrix. In contrast to classical MDS and Isomap, however, the content of that matrix is subject to a preceding optimization problem, which can be cast as a semidefinite program.

In particular, the goal of SDE/MVU is to find the embedding $\{\mathbf{x}_i\}$ which has maximal variance (cf. PCA), under the constraints that the embedding is centered, and local distances and angles are preserved. By introducing elements $k_{ij} = \mathbf{x}_i \cdot \mathbf{x}_j$ of a dot product matrix $\mathbf{K}$, the variance of the embedding is proportional to the trace of $\mathbf{K}$, and the centering constraint is equivalent to demanding $\sum_{ij} k_{ij} = 0$. If we further introduce the elements

$$a_{ij} = \begin{cases} 1 & \mathbf{y}_i \text{ and } \mathbf{y}_j \text{ are neighbors} \\ 0 & \text{otherwise} \end{cases} \tag{2.83}$$

of an adjacency matrix $\mathbf{A}$, the distance and angle preservation constraints can be written

as (Weinberger and Saul, 2004)

$$k_{ii} - 2k_{ij} + k_{jj} = \|\mathbf{y}_i - \mathbf{y}_j\|^2 \ , \text{ if } a_{ij} = 1 \text{ or } (\mathbf{A}^T\mathbf{A})_{ij} > 0. \tag{2.84}$$

Here, local angles are preserved by demanding that if $\mathbf{y}_k$ and $\mathbf{y}_j$ are neighbors of $\mathbf{y}_i$, then also the distance $\|\mathbf{y}_k - \mathbf{y}_j\|$ has to be preserved.

Please note that the complete optimization problem can be written in terms of $\mathbf{K}$, and that the objective function and the constraints are either linear in k_{ij} (trace, centering, preservation), or positive semidefinite ($\mathbf{K}$ being a dot product matrix). The optimization problem is thus convex, and its unique solution can be retrieved by semidefinite programming. When the optimal $\mathbf{K}$ has been found, matching coordinates $\{\mathbf{x}_i\}$ are retrieved by an eigendecomposition, where the magnitude of the eigenvalues indicate the necessary dimensionality.

SDE/MVU can also be viewed as a form of Kernel PCA, where the kernel is not specified in advance, but the kernel matrix is learned. A landmark variant (ℓSDE), which speeds up the algorithm by factorizing the kernel matrix, has been introduced by Weinberger, Packer, and Saul (2005).

2.8.4. Further methods and discussion

Laplacian Eigenmaps, as introduced by Belkin and Niyogi (2002, 2003), share similarities with both Isomap and LLE. The algorithm is based on minimizing an error measure similar to the general MDS error (2.73), with weights w_{ij} given by a "heat kernel" with a "temperature" paramter t (Belkin and Niyogi, 2002)

$$w_{ij} = \begin{cases} \exp\left(-\frac{1}{t}\|\mathbf{y}_i - \mathbf{y}_j\|^2\right) & \text{if } \mathbf{y}_i \text{ and } \mathbf{y}_j \text{ are neighbors} \\ 0 & \text{otherwise.} \end{cases} \tag{2.85}$$

Similar to LLE, the optimal embedding is retrieved by minimizing the trace of $\mathbf{XLX}^T$ by eigendecomposition of a sparse matrix $\mathbf{L}(\mathbf{W})$, which the authors identify with a Laplacian operator on the graph.

Another similar algorithm is HLLE, or Hessian Eigenmaps (Donoho and Grimes, 2003), which can be coarsely described as Laplacian Eigenmaps with the Laplacian operator replaced by the Hessian. HLLE involves an estimation of the tangent space of the manifold, which yields an accurate description of the local geometric structure, but which is sensitive to noise. In contrast to LLE, Isomap, and Laplacian Eigenmaps, both HLLE and MVU can handle manifolds whose intrinsic structure is not convex.

As a brief summary, spectral methods retrieve the optimal embedding by an eigendecomposition of an $N \times N$ matrix. In case of LLE, Laplacian Eigenmaps, and HLLE, this matrix is sparse, and the interesting eigenvalues lie at the bottom of the spectrum. In contrast, Isomap and MVU seek the largest eigenvalues of a non-sparse matrix, which is costlier, but in both cases the magnitude of the eigenvalues can be utilized to determine the right dimensionality of the embedding. In case of MVU, a semidefinite program has to be solved as an additional step. A detailed discussion of spectral embedding algorithms and their relation to Kernel PCA has been provided by Ham, Lee, Mika, and Schölkopf (2004), Saul, Weinberger, Ham, Sha, and Lee (2006), and Bengio, Delalleau, Paiement, Vincent, and Ouimet (2004).

All aforementioned methods depend on a neighborhood parameter K or ϵ, which can dramatically influence the shape of the final embedding, as will be demonstrated later in chapter 4. A major problem for the selection of the neighborhood parameter are so-called shortcut connections, which are illustrated in Fig. 2.15.

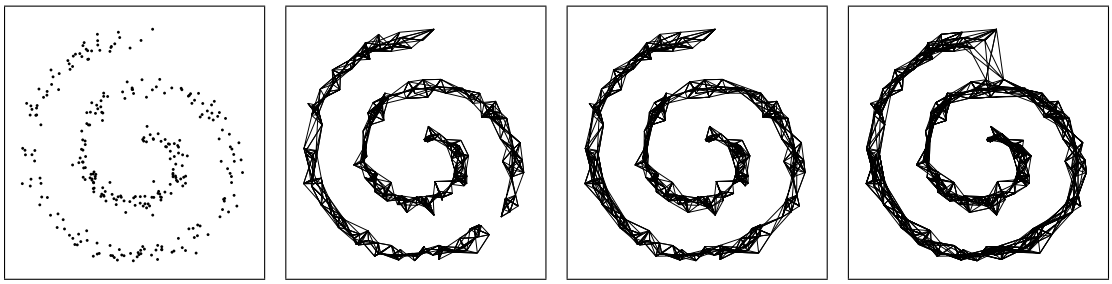


Figure 2.15: Illustration of the neighborhood graph and its dependency on the parameter K. From left to right: original dataset, neighborhood graph for $K = 6$, $K = 7$, and $K = 9$. While $K = 6$ does not yield a connected graph (thus preventing a contiguous embedding), $K = 9$ shows short-cut connections, which will heavily disturb the embedding process.

Several research papers provide a comparison of nonlinear spectral embedding methods to the class of iterative algorithms described in the preceding section. For example, Isomap has been compared to CDA by Lee et al. (2004), and Lee, Archambeau, and Verleysen (2003) compared LLE with the Isotop algorithm (Lee and Verleysen, 2002). To summarize their argumentation, the mathematical and theoretical elegance of spectral embedding algorithms can hardly be beaten, but with a careful tuning of their learning parameters, iterative algorithms show more robustness and are able to cope with a broader class of problems.

While spectral methods heavily rely on an implicit manifold model, like the methods of the preceding section, they do not feature a functional relationship $\mathbf{y} = \mathbf{f}(\mathbf{x})$. Of course, once the algorithm is run, one could use any supervised regression technique to create such a mapping, but this would yield two independent steps, where the involved models cannot benefit from each other. An extension of spectral methods that permits the embedding of new data points (similar to CCA, Sec. 2.7.3) has been proposed by Bengio, Paiement, et al. (2004). Note however, that such an embedding is not the same as a projection of data onto an explicit manifold model, as is possible e.g. with PCA or GTM. For example, one cannot calculate the distance between a data vector $\mathbf{y}$ and the manifold spanned by $\mathbf{f}(\cdot)$.

3. The Parametrized Self-Organizing Map and extensions

The Parametrized Self-Organizing Map (PSOM) has been introduced by Ritter (1993) as a continuous generalization of the standard SOM (cf. Sec. 2.4). It inherits the SOM's ability to create topology preserving mappings between data points in a possibly high-dimensional space $\mathbb{R}^d$ and the nodes of a Cartesian grid in a lower dimensional space $\mathbb{R}^q$, but extends the SOM by explicitly defining a continuous manifold in the data space through a smooth mapping $\mathbf{f} : \mathbb{R}^q \to \mathbb{R}^d$.

While the PSOM was designed to operate on the weights $\{\mathbf{w}_j\}$ of a readily trained SOM, such pre-processing is not always needed. Often the training data (that is, the weights) can already be generated by sampling along the degrees of freedom of a system, for example in learning the forward and inverse kinematics of a robot arm (Ruiz de Angulo and Torras, 2002; Padoan Jr., Barreto, and Araújo, 2003) or of single fingers (Nölker and Ritter, 2000; Walter, Nölker, and Ritter, 2000) and in object recognition and pose estimation (Saalbach, Heidemann, and Ritter, 2002). By including topology information, the PSOM permits the construction of highly accurate mappings from very few training examples (Walter, 1998).

However, in its original form the PSOM suffers from two restrictions. Firstly, the algorithm features no explicit consideration of noise that might be present in the data (e.g. in physical measurements). Secondly, the original PSOM requires a complete set of grid-organized data to construct its mapping. This means that (i) training data lying in between the grid nodes can not be incorporated and (ii) even a single missing weight (e.g. a sample position not realizable because of physical constraints) makes the PSOM construction algorithm inapplicable.

This chapter of the thesis describes how these two issues can be addressed by an elegant integration of standard smoothing techniques into the PSOM framework. Specifically, we present an approach to regularize PSOM mappings in order to deal with noisy data in a principled manner and we provide a modification of the original algorithm, allowing to construct PSOMs from data that are not organized in a grid topology. The latter includes grid-based data with missing elements as a special case.

In accordance with the original publication (Klanke and Ritter, 2005) of these modifications, we will use the notation PSOM$^+$ when we wish to explicitly indicate the application of the new regularization approach.

Our method is based on measuring the overall smoothness of the PSOM mapping $\mathbf{f}(\cdot)$. In particular, we integrate the square sum of all second derivatives of $\mathbf{f}(\cdot)$, the result of which can be expressed by a quadratic form. As a consequence, the problem of finding optimal (here: maximally smooth) PSOM$^+$ mappings can be solved by applying linear algebra.

In the next section, we briefly recall the original PSOM algorithm and introduce some

necessary notation. After that, we illustrate how the PSOM can be applied in a kinematics learning task, and thereby motivate our extensions to the PSOM framework. Then, we derive a metric in the space of PSOM weights, which allows us to calculate the overall smoothness of a PSOM manifold as a function of its weights. We show how to tackle the problems of noisy data, missing weights and non grid-organized data and illustrate our method by toy examples. After that, we return to the (simulated) kinematics learning task, where we demonstrate the performance of the PSOM$^+$ extensions. Finally, we indicate how to utilize the PSOM$^+$ in an unsupervised setting without a preceding SOM training, and we conclude this chapter with a discussion.

3.1. Original formulation

As described in section 2.4, a conventional SOM consists of an array of formal neurons arranged on the nodes of a q-dimensional grid. Each neuron, which we characterize by a multi-index $\mathbf{g} = (g^1, g^2, \ldots, g^q)^T$, has a d-dimensional weight $\mathbf{w_g}$ attached. The entirety of the weights, along with their respective neuron topology within the grid, forms a discrete approximation of a possibly nonlinear manifold embedded in $\mathbb{R}^d$.

The PSOM is built on the same kind of neuron array, but interpolates the weights by a smooth vector-valued function $\mathbf{f}(\mathbf{x})$. The range of that function defines a manifold parameterized by a continuous q-dimensional quantity, the coordinate $\mathbf{x}$ within the grid. The relation between a SOM and a corresponding PSOM is illustrated in Fig. 3.1.

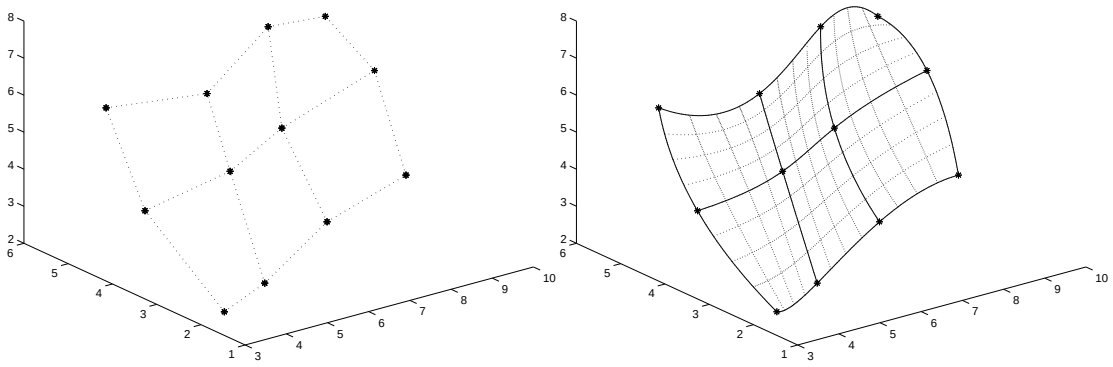


Figure 3.1: From SOM to PSOM. The left plot shows a 4x3 SOM with 3D weights, the right plot depicts the corresponding PSOM.

While the SOM algorithm has been utilized in conjunction with a wide variety of shapes of the underlying grid (e.g. hexagonal or ring-shaped lattices), in the following, we only work on Cartesian (but not necessarily regular) grids. In this case, the set of node coordinates $\mathbf{A} = \{\mathbf{a_g}\}$ is given by a Cartesian product of 1-D coordinate sets A^μ along the different dimensions[1]:

$$\mathbf{A} = A^1 \times A^2 \times \cdots \times A^q, \quad A^\mu = \{a_1^\mu, a_2^\mu, \ldots, a_{n^\mu}^\mu\}, \quad \mu = 1 \ldots q. \tag{3.1}$$

[1] Throughout the chapter, we denote the grid dimension by an upper Greek index. In contrast to the SOM formulation in Sec. 2.4, the grid coordinates of the nodes are denoted by $\mathbf{a}_*$.

Here, n^μ denotes the number of nodes along grid dimension μ. The PSOM mapping is given by multi-dimensional Lagrange interpolation (e.g. de Boor, 1978), that is

$$\mathbf{f}(\mathbf{x}) = \sum_{\mathbf{g}} \mathbf{w}_{\mathbf{g}} b_{\mathbf{g}}(\mathbf{x}) = \sum_{g^1=1}^{n^1} \sum_{g^2=1}^{n^2} \cdots \sum_{g^q=1}^{n^q} \mathbf{w}_{\mathbf{g}} \prod_{\mu=1}^{q} l_{g^\mu}^\mu(x^\mu), \tag{3.2}$$

where $\mathbf{x} = (x^1, x^2, \ldots, x^q)^T$ and $l_i^\mu(x^\mu)$ are standard one-dimensional Lagrange polynomials

$$l_{g^\mu}^\mu(x^\mu) = \prod_{\substack{j=1 \\ j \neq g^\mu}}^{n^\mu} \frac{x^\mu - a_j^\mu}{a_{g^\mu}^\mu - a_j^\mu}. \tag{3.3}$$

It is easy to see that the PSOM mapping $\mathbf{f}(\mathbf{x})$ exactly interpolates the SOM weights, that is, if one evaluates (3.2) at a node position $\mathbf{a}_\mathbf{h}$, one gets $\mathbf{f}(\mathbf{a}_\mathbf{h}) = \mathbf{w}_\mathbf{h}$. Concerning this, note that

$$l_{g^\mu}^\mu(a_{h^\mu}^\mu) = \prod_{\substack{j=1 \\ j \neq g^\mu}}^{n^\mu} \frac{a_{h^\mu}^\mu - a_j^\mu}{a_{g^\mu}^\mu - a_j^\mu} = \begin{cases} 1 & \text{if } h^\mu = g^\mu \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

and thus

$$b_{\mathbf{g}}(\mathbf{a}_\mathbf{h}) = \prod_{\mu=1}^{q} l_{g^\mu}^\mu(a_{h^\mu}^\mu) = \begin{cases} 1 & \text{if } \mathbf{h} = \mathbf{g} \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

Inserting (3.5) into (3.2) then yields $\mathbf{w}_\mathbf{h}$.

As a consequence of the continuous definition, the PSOM differs from the SOM also in the process of projecting a data vector $\mathbf{y} \in \mathbb{R}^d$ onto the manifold. While the standard SOM responds to a data vector $\mathbf{y}$ by the nearest weight $\mathbf{w}_{\mathbf{g}^*}$ ("discrete best match"), the PSOM responds by $\mathbf{f}(\mathbf{x}^*)$ with $\mathbf{x}^*$ given as the solution of the continuous minimization problem

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} d(\mathbf{f}(\mathbf{x}), \mathbf{y}), \tag{3.6}$$

where $d(\mathbf{y}, \mathbf{y}')$ may e.g. be chosen as the standard Euclidean distance $\|\mathbf{y} - \mathbf{y}'\|$. By including only certain components of $\mathbf{y}$ and $\mathbf{f}(\mathbf{x})$ in the best match search, that is, using a distance function like

$$d(\mathbf{y}, \mathbf{y}') = \sum_{i=1}^{d} p_i (y_i - y_i')^2, \tag{3.7}$$

where some $p_i = 0$, the PSOM can be used as a "continuous associative memory" or as a "multi-map" tool, for example to unite the forward and inverse kinematics of a robot in one mapping (Ruiz de Angulo and Torras, 2002; Nölker and Ritter, 2000).

3.1.1. Chebyshev PSOMs and local PSOMs

As described above, the function model of the PSOM is a product of 1-D polynomials. If the PSOM has n^μ nodes along grid dimension μ, the corresponding polynomial is of degree $n^\mu - 1$. Thus, if the grid of the PSOM is large, the degree of the polynomials is large, too. This can raise severe problems near the borders of the grid, since polynomials of high degree are famous for showing oscillatory behavior and bad extrapolation

performance (de Boor, 1978). If a global polynomial model is used to approximate a function from samples $\{(x_i, y_i)\}$, the location of the sample points $\{x_i\}$ have a strong influence on the quality of the approximation. Contrary to intuition, regularly sampling an interval (equidistant sample points) does *not* yield optimal results.

A much better strategy is to place the sampling points at the zeros $\{a_i\}$ of the n-th degree Chebyshev polynomial, given by[2]

$$a_i = -\cos\left(\pi \frac{i - \frac{1}{2}}{n}\right), \quad i = 1 \ldots n. \tag{3.8}$$

It can be shown that Chebyshev-spaced sample points are the best choice, and that the resulting polynomial is nearly optimal among all approximations by a polynomial of degree n (de Boor, 1978). As an illustration, Fig. 3.2 depicts the results of approximating the function $g(x) = \exp(-8x^2)$ by polynomial interpolation of 7 and 12 sampling points, with Chebyshev- and regular spacing.

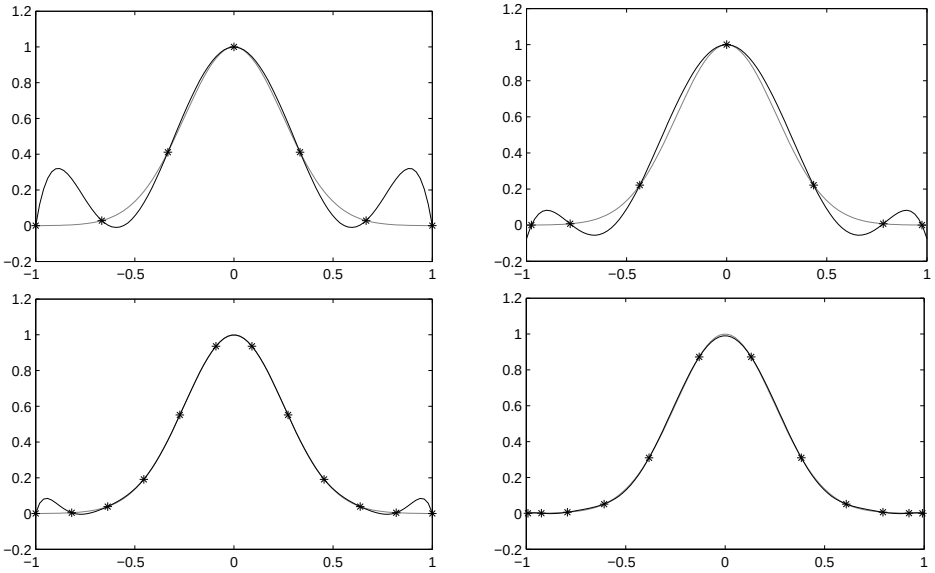


Figure 3.2: Approximation of a "Gaussian bell" by polynomials. Left column: regular spacing, right column: Chebyshev spacing. Top row: 7 sample points, bottom row: 12 sample points. Note the much better behavior at the borders for the Chebyshev-spaced polynomials.

Utilizing a Chebyshev-spacing for the nodes of a PSOM grid was proposed by Walter and Ritter (1995), who also introduced the concept of a *local* PSOM. The latter works by restricting the Lagrangian interpolation to a dynamically selected sub-grid with for example $n_l = 3$ nodes per dimension. The resulting local PSOM thus features a simplified function model, built upon 1-D polynomials with degree $n_l - 1$. This both prevents heavy oscillations and reduces the computational burden as compared to the global PSOM. As a disadvantage, neighboring local PSOMs with overlapping sub-grids do not yield the same sub-manifold at the overlap (cf. Fig. 3.3). A related, but more

[2]Here we assume that the approximation should hold in the interval $[-1; 1]$. In general the sample points have to be shifted and scaled accordingly.

complex technique for local interpolation is based on tensor products of splines (Green and Silverman, 1994).

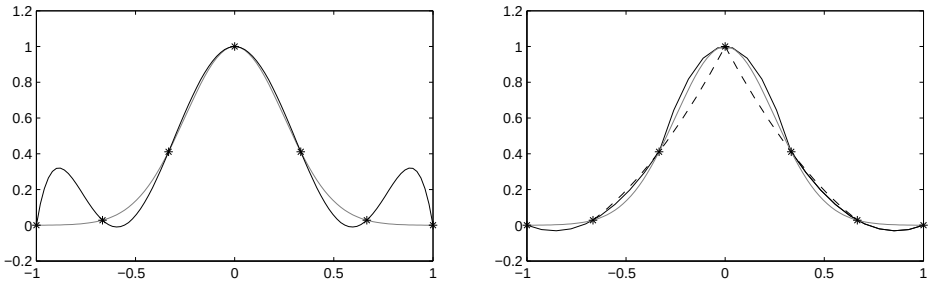


Figure 3.3: Approximation of a "Gaussian bell" by polynomials. Left: regularly spaced full PSOM, 7 nodes (cf. Fig. 3.2). Right: local PSOMs based on 3 nodes each. The local PSOMs on nodes 1–3, 3–5, and 5–7 are depicted by a solid black curve, the PSOMs on nodes 2–4 and 4–6 are drawn as a dashed curve. Note the deviation between local PSOMs at overlapping regions.

3.2. Application in kinematics learning

As already mentioned above, the PSOM algorithm has often been applied in robotics, particularly in learning the kinematics of a robot arm. In such a setting, the weight vectors often do not have to be determined from a preceding SOM training, but can be generated by sampling. In this section, we utilize the PSOM to learn a part of the kinematics of the Mitsubishi PA-10 robot arm. The PA-10, depicted in Fig. 3.4, features 7 joints and thus is a redundant manipulator: a desired end-effector position and orientation can be specified by 6 physical values (e.g. by 3 Euler angles and the position in $\mathbb{R}^3$), but the robot has 7 degrees of freedom (DOF), namely the joint angles $\boldsymbol{\Theta} = (\Theta_1, \Theta_2, \ldots, \Theta_7)$.

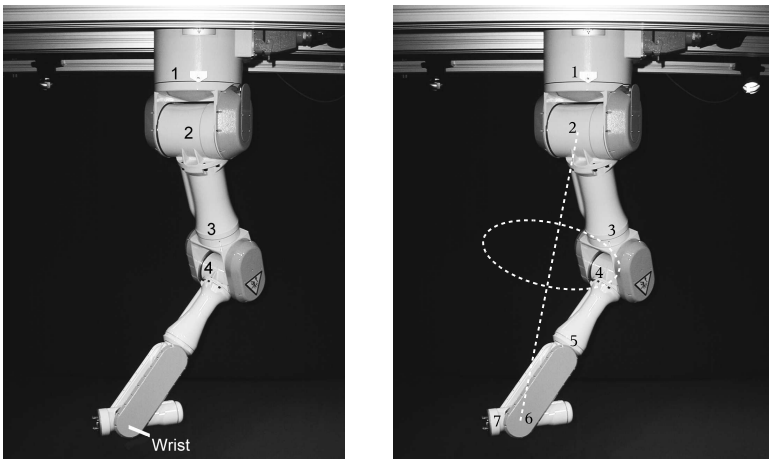


Figure 3.4: Mitsubishi PA-10. Left: the position of the wrist depends only on the first 4 joint angles. Right: any position of the wrist can be realized by infinitely many postures of the arm, where the center of joint 4 moves along the depicted circle.

As described for example by Walter (1998), one can build a PSOM model for the kinematics by specifying a multi-dimensional grid in the space of DOFs, moving the robot to every node posture $\mathbf{\Theta_g}$, and measuring the resulting position $\mathbf{r_g}$ and orientation $\mathbf{o_g}$. The weight vectors $\mathbf{w_g}$ at nodes $\mathbf{g}$ are then composed of the corresponding measurements. Unfortunately, a single PSOM for the complete PA-10 kinematics is not practical, because the number of nodes rises exponentially with the dimensionality of the grid, and the cost of evaluating the PSOM function $\mathbf{f}(\mathbf{x})$ is linear in that number. Even if we sample at only 5 locations per DOF, the resulting PSOM would consist of $5^7 = 78,125$ nodes.

To address this problem, one can decompose the kinematics[3] of the arm and utilize independent PSOMs for the sub-models (Ruiz de Angulo and Torras, 2002). In case of the PA-10, a suitable decomposition[4] is given by handling the first 4 and the last 3 joints separately: Given a desired end-effector position and orientation, the corresponding position of the wrist is unique and already fixed, and it depends only on the first 4 joint angles (cf. Fig. 3.4). Note that the problem is still redundant.

In the following, we describe how to build a PSOM for the mapping between the joint angles $\mathbf{\Theta} = (\Theta_1, \Theta_2, \Theta_3, \Theta_4)^T$ and the wrist position $\mathbf{r} = (r_x, r_y, r_z)^T$. At first, we specify a 4-D grid in the space of joint angles, such that each grid axis covers the range of one joint. Let n^μ denote the number of nodes along dimension μ, and $[\Theta_\mu^{min}; \Theta_\mu^{max}]$ the range of the μ-th joint, than the node positions for a regularly spaced PSOM are given by

$$a_i^\mu = \Theta_\mu^{min} + \frac{i-1}{n^\mu - 1}(\Theta_\mu^{max} - \Theta_\mu^{min}) \ , \quad i = 1 \ldots n^\mu, \tag{3.9}$$

whereas the node coordinates of a Chebyshev PSOM are

$$a_i^\mu = \Theta_\mu^{min} + \frac{1}{2}\left(1 - \cos\left(\pi \frac{i - \frac{1}{2}}{n}\right)\right)(\Theta_\mu^{max} - \Theta_\mu^{min}) \ , \quad i = 1 \ldots n^\mu. \tag{3.10}$$

While the PSOM mapping is invariant under scaling and translation of the node coordinates, by this construction the 4-D grid coordinates $\mathbf{a_g}$ can directly be utilized as the joint angles $\mathbf{\Theta_g}$ corresponding to that node. For every node $\mathbf{g}$, we then move the robot to the posture given by $\mathbf{\Theta_g} = \mathbf{a_g}$ and measure the position $\mathbf{r_g}$ (and optionally also the orientation) of the wrist. If we now simply set $\mathbf{w_g} = \mathbf{r_g}$, the PSOM mapping $\mathbf{f} : \mathbf{x} \in \mathbb{R}^4 \rightarrow \mathbf{y} \in \mathbb{R}^3$ approximates the forward kinematics $\mathbf{r} = \mathbf{f}(\mathbf{\Theta})$.

More interesting mappings can be created if also the joint angles are put into the weight vectors, yielding a 7-D vector $\mathbf{w} = (r_x, r_y, r_z, \Theta_1, \Theta_2, \Theta_3, \Theta_4)^T$. Now, the PSOM mapping describes the kinematics as a 4-D manifold embedded in $\mathbb{R}^7$. As described by Walter (1998) for a 3-DOF robotic finger, we can utilize the PSOM manifold as a flexible inverse kinematics solver:

- Imagine the current joint angles are $\mathbf{\Theta}_0$, and the robot should be moved to wrist position $\mathbf{r}_{dest}$. Then we set $\mathbf{y} = (\mathbf{r}_{dest}^T, 0, 0, 0, 0)^T$ and simply carry out the continuous best-match search (3.6), starting from $\mathbf{x} = \mathbf{\Theta}_0$, where we set $p_1 = p_2 = p_3 = 1$

[3] For a high number of DOFs, or high-dimensional input spaces in general, an alternative learning algorithm has been proposed by Vijayakumar and Schaal (2000). Instead of learning decomposed models globally, multiple local models are combined similarly to local PCA.

[4] On a related note, a similar decomposition strategy has also been succesfully applied to a grid-based path planning algorithm (Klanke, Lebedev, Haschke, Steil, and Ritter, 2006).

and $p_4 = p_5 = p_6 = p_7 = 0$ in the distance measure (3.7). Because of the redundancy, there are multiple possible solutions for the joint angles, but e.g. a steepest-descent procedure will yield a minimum which is close to the starting point $\mathbf{\Theta}_0$.

- We could additionally demand that e.g. Θ_2 has a certain value Θ_2^*, and solve the inverse kinematics task by setting $\mathbf{y} = (\mathbf{r}_{dest}^T, 0, \Theta_2^*, 0, 0)^T$ and changing p_5 to 1. If no solution with $d(\mathbf{y}, \mathbf{f}(\mathbf{x})) = 0$ is found, we can repeat the procedure with smaller values for p_5 to balance between the joint angle constraint and the deviation from the target position.

- One could also include further measurements into the weight vectors, which for example describe how "convenient" a robot posture is, and adjust the influence of such measurements on the best match search through $\{p_i\}$.

These multi-map properties of the PSOM, however, are not the focus of this thesis, but rather the accuracy of the PSOM and its requirements on the form of the training data.

In order to demonstrate the approximation accuracy of the PSOM algorithm, we created 8x8x8x8 PSOMs with both regular and Chebyshev spacing, and compared the PSOM mappings to the true analytic forward kinematics, which we also used to calculate the 4096 weight vectors $\mathbf{w_g} = \mathbf{r_g}(\mathbf{\Theta_g})$. We evaluated the performance on 1000 randomly generated postures (joint angle sets). The results for the full PSOM, local PSOMs of sub-grid size 3 and 5, and also for a cubic spline interpolation[5] are depicted in Table 3.1. Note the very small mean positional error for the full PSOM, particularly on the Chebyshev grid. In relation to the arm length of the PA-10 ($\approx$ 1.30m), the mean error is well under 0.1%.

	Deviation of wrist position in mm (mean ± standard deviation)	
Method	Regular spacing	Chebyshev spacing
Full PSOM	0.80 ± 0.86	**0.54 ± 0.27**
Local PSOM ($n_l = 3$)	17.92 ± 9.35	34.95 ± 16.67
Local PSOM ($n_l = 5$)	3.59 ± 2.48	18.01 ± 7.47
Cubic splines	3.59 ± 3.25	2.56 ± 1.47

Table 3.1: Accuracy of approximating the forward kinematics. Joint angles and corresponding wrist positions for the training data were sampled at the nodes of a 8x8x8x8 grid. The performance was evaluated on 1000 randomly generated postures.

This accuracy, however, comes at a price. The training data has to be highly structured, i.e. a wrist position must be available at every node of the grid. If the forward kinematics is not known analytically, we actually have to move the robot to every node posture, which may not be possible because of workspace constraints. A similar argument was listed by Ruiz de Angulo and Torras (2002) in support of their kinematics decomposition scheme, but of course the PSOM's dependance on a complete grid-ordered training set is a major obstruction in other applications, as well.

[5]The cubic spline interpolation was calculated using the MATLAB function `interpn`.

Another major drawback of the PSOM is its sensitivity to noise. If in our example the measurements $\mathbf{r_g}$ are not exact, the PSOM will actually interpolate the errors as well.

3.3. PSOM$^+$ extensions

In this section, we describe our extensions to the PSOM algorithm, which are aimed to overcome the aforementioned drawbacks. We restrict ourselves to full (i.e. not local) PSOMs, and we first derive an explicit measure of the overall smoothness of the mapping.

3.3.1. Explicit smoothness measure

A frequently used measure of smoothness, or rather roughness, of a function $f : \mathcal{D} \subset \mathbb{R} \rightarrow \mathbb{R}$ is given by the integral over its squared second derivative

$$R(f) = \int_{\mathcal{D}} \left(f''(x)\right)^2 dx. \tag{3.11}$$

The theory of cubic smoothing splines, for example, is based on this measure (Green and Silverman, 1994, also cf. Sec. 2.5.1).

For multivariate functions like the PSOM mapping, the integration has to take all second derivatives into account, but the different output components can be treated separately. Therefore, in the following, we view the PSOM weights as one-dimensional. The range of integration matches the hyper-rectangle spanned by the grid. For a fixed grid, the PSOM basis functions $b_{\mathbf{g}}(\mathbf{x})$ are also fixed, and thus the roughness of the PSOM mapping depends exclusively on the weights:

$$\begin{aligned}
R(\{w\}) &= \int_{\mathcal{D}} \sum_{\mu=1}^{q} \sum_{\nu=1}^{q} \left(\frac{\partial^2}{\partial x^\mu \partial x^\nu} f(\mathbf{x}) \right)^2 d^q\mathbf{x} && (3.12)\\
&= \sum_{\mu,\nu} \int_{\mathcal{D}} \left(\sum_{\mathbf{g}} w_{\mathbf{g}} \frac{\partial^2}{\partial x^\mu \partial x^\nu} b_{\mathbf{g}}(\mathbf{x}) \right)^2 d^q\mathbf{x} && (3.13)\\
&= \sum_{\mathbf{g},\mathbf{h}} w_{\mathbf{g}} w_{\mathbf{h}} \sum_{\mu,\nu} \int_{\mathcal{D}} \left(\frac{\partial^2}{\partial x^\mu \partial x^\nu} b_{\mathbf{g}}(\mathbf{x}) \right) \left(\frac{\partial^2}{\partial x^\mu \partial x^\nu} b_{\mathbf{h}}(\mathbf{x}) \right) d^q\mathbf{x} && (3.14)\\
&= \sum_{\mathbf{g},\mathbf{h}} w_{\mathbf{g}} w_{\mathbf{h}} \sum_{\mu,\nu} I^{\mu\nu}_{\mathbf{gh}} = \sum_{\mathbf{g},\mathbf{h}} w_{\mathbf{g}} w_{\mathbf{h}} M_{\mathbf{gh}}. && (3.15)
\end{aligned}$$

Here, we used the definition

$$I^{\mu\nu}_{\mathbf{gh}} = \int_{\mathcal{D}} \left(\frac{\partial^2}{\partial x^\mu \partial x^\nu} b_{\mathbf{g}}(\mathbf{x}) \right) \left(\frac{\partial^2}{\partial x^\mu \partial x^\nu} b_{\mathbf{h}}(\mathbf{x}) \right) d^q\mathbf{x}. \tag{3.16}$$

Since the basis functions $b_{\mathbf{g}}(\mathbf{x})$ are just products of one-dimensional polynomials, their derivatives are quite simple and products of one-dimensional polynomials as well. De-

noting first and second derivatives by $'$ and $''$, we get

$$\frac{\partial^2}{\partial x^\mu \partial x^\nu} b_{\mathbf{g}}(\mathbf{x}) = \begin{cases} \prod_{\alpha \neq \mu} l_g^\alpha(x^\alpha) l_g^{\mu''}(x^\mu) & \mu = \nu \\ \prod_{\alpha \neq \mu,\nu} l_g^\alpha(x^\alpha) l_g^{\mu'}(x^\mu) l_g^{\nu'}(x^\nu) & \mu \neq \nu. \end{cases} \tag{3.17}$$

To keep the notation compact, we omitted the double index of g, that is, l_g^α is to be read as an abbreviation of $l_{g^\alpha}^\alpha$. We also omitted brackets around product terms like $\prod_\alpha z_\alpha$. Further simplification of (3.16) requires a case distinction and to separate the terms depending on their dimension indices. For the first case ($\mu = \nu$) one gets

$$I_{\mathbf{gh}}^{\mu\mu} = \int \prod_{\alpha \neq \mu} l_g^\alpha(x^\alpha) l_g^{\mu''}(x^\mu) \prod_{\beta \neq \mu} l_h^\beta(x^\beta) l_h^{\mu''}(x^\mu) d^q\mathbf{x} \tag{3.18}$$

$$= \prod_{\alpha \neq \mu} \underbrace{\int l_g^\alpha(x^\alpha) l_h^\alpha(x^\alpha) dx^\alpha}_{\mathcal{A}_{gh}^\alpha} \underbrace{\int l_g^{\mu''}(x^\mu) l_h^{\mu''}(x^\mu) dx^\mu}_{\mathcal{B}_{gh}^\mu}, \tag{3.19}$$

while the second case ($\mu \neq \nu$) yields

$$I_{\mathbf{gh}}^{\mu\nu} = \int \prod_{\substack{\alpha \neq \mu \\ \alpha \neq \nu}} l_g^\alpha(x^\alpha) l_g^{\mu'}(x^\mu) l_g^{\nu'}(x^\nu) \prod_{\substack{\beta \neq \mu \\ \beta \neq \nu}} l_h^\beta(x^\beta) l_h^{\mu'}(x^\mu) l_h^{\nu'}(x^\nu) d^q\mathbf{x} \tag{3.20}$$

$$= \prod_{\substack{\alpha \neq \mu \\ \alpha \neq \nu}} \underbrace{\int l_g^\alpha(x^\alpha) l_h^\alpha(x^\alpha) dx^\alpha}_{\mathcal{A}_{gh}^\alpha} \underbrace{\int l_g^{\mu'}(x^\mu) l_h^{\mu'}(x^\mu) dx^\mu}_{\mathcal{C}_{gh}^\mu} \underbrace{\int l_g^{\nu'}(x^\nu) l_h^{\nu'}(x^\nu) dx^\nu}_{\mathcal{C}_{gh}^\nu}. \tag{3.21}$$

So, all we need to calculate are the symmetric matrices $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$ for each grid dimension. A convenient way to calculate the integrals is to first build a coefficient representation of the polynomials $l_g^\alpha(x^\alpha)$, which makes differentiation and integration very simple. The remaining work consists of collecting the proper summands for the matrix elements $M_{\mathbf{gh}} = \sum_{\mu,\nu} I_{\mathbf{gh}}^{\mu\nu}$.

In the following, we drop the multi-index notation in favor of a binary representation of the indices, and write $M_{\mathbf{gh}}$ and $w_{\mathbf{g}}$ in matrix and vector notation $\mathbf{M}$ and $\mathbf{w}$, through which the roughness measure (3.15) can be expressed by

$$R(\mathbf{w}) = \mathbf{w}^T \mathbf{M} \mathbf{w}. \tag{3.22}$$

For a grid with $N = \prod_{\mu=1}^q n^\mu$ nodes, $\mathbf{M}$ is an $N \times N$ matrix, while the weights are represented by a vector $\mathbf{w} \in \mathbb{R}^N$ for each dimension of the data space. Please note that the matrices $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$ are much smaller, i.e. for grid dimension α they have only $n^\alpha \times n^\alpha$ elements. The matrix $\mathbf{M}$ depends only on the placement of the nodes and not on the weights, and therefore it has to be calculated only once for a given grid layout. Note that by construction, $\mathbf{M}$ defines a symmetric positive semidefinite metric in the space $\mathbb{R}^N$ of weights.

We are now ready to utilize the metric $\mathbf{M}$ in order to tackle the problems stated above. We begin with showing how to make use of our roughness measure $R(\{w\})$ to construct PSOM$^+$ mappings from noisy data. For an illustration of the approach, we apply it to 1-D toy data.

3.3.2. Noisy data

Suppose that the training data $\tilde{\mathbf{w}} = (\tilde{w}_g)$ is complete (each node has a weight vector), but noisy. For the case of Gaussian noise, this can be stated as

$$\tilde{w}_g = \hat{w}_g + u \qquad u \sim \mathcal{N}(0, \sigma^2), \tag{3.23}$$

where $\hat{\mathbf{w}} = (\hat{w}_g)$ denotes the "true" weights. The less we can trust the data, the smoother the PSOM$^+$ mapping should be, so a good strategy is to use slightly modified ("de-noised") weights $\mathbf{w}$ that minimize

$$\begin{aligned} E(\mathbf{w}, \lambda) &= \|\mathbf{w} - \tilde{\mathbf{w}}\|^2 + \lambda R(\mathbf{w}) && (3.24) \\ &= (\mathbf{w} - \tilde{\mathbf{w}})^T(\mathbf{w} - \tilde{\mathbf{w}}) + \lambda \mathbf{w}^T \mathbf{M} \mathbf{w}. && (3.25) \end{aligned}$$

Here, λ acts as a regularization parameter that balances between a smaller modification of the weights and a smoother resulting PSOM$^+$ mapping. The optimal weights $\mathbf{w}_{opt}$ can be found by solving

$$\mathbf{0} \stackrel{!}{=} \frac{\partial E_\lambda}{\partial \mathbf{w}} = 2(\mathbf{w} - \tilde{\mathbf{w}}) + 2\lambda \mathbf{M} \mathbf{w}, \tag{3.26}$$

which yields

$$\mathbf{w}_{opt} = (\mathbf{I} + \lambda \mathbf{M})^{-1} \tilde{\mathbf{w}}. \tag{3.27}$$

If an estimate of the noise variance σ^2 is available, we can use a simple heuristics and select the regularization parameter by adjusting λ such that $\frac{1}{N}\|\mathbf{w}_{opt} - \tilde{\mathbf{w}}\|^2 \approx \sigma^2$. In this way, we seek a modification of $\tilde{\mathbf{w}}$ that is roughly as strong as the corruption of $\hat{\mathbf{w}}$ by noise.

Figure 3.5 illustrates the result of (3.27) on toy data for the case of a one-dimensional PSOM$^+$ with 8 nodes spaced at $a_i = i$, $(i = 1 \dots 8)$.

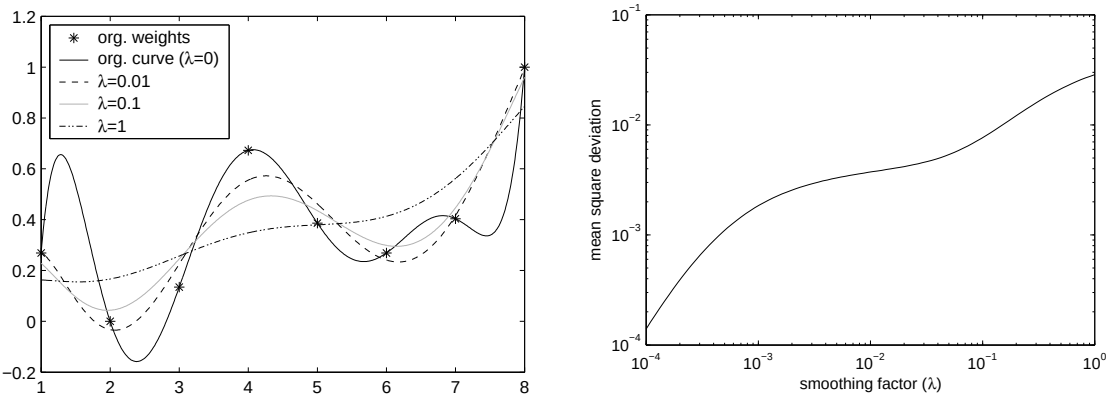


Figure 3.5: PSOM$^+$ from noisy data. Left: resulting mappings for different settings of the regularization parameter λ. Right: mean square distance $\frac{1}{N}\|\mathbf{w}_{opt} - \tilde{\mathbf{w}}\|^2$ between de-noised and original weights as a function of the smoothing parameter.

3.3.3. Missing data

Now suppose the training data to be free of noise, but instead incomplete. Given a set of indices to which the weights are known, and a complementary unknown set, what is now the optimal choice for the unknown weights?

The less you know about a function, the simpler your estimate of it should be, so again we choose the missing weights in a way that maximizes smoothness (minimizes roughness). Denoting sub-matrices (sub-vectors) of $\mathbf{M}$ and $\mathbf{w}$ by indices u (unknown) and k (known), the roughness measure can be expressed as

$$R(\mathbf{w}) = \mathbf{w}_k^T \mathbf{M}_{kk} \mathbf{w}_k + \mathbf{w}_k^T \mathbf{M}_{ku} \mathbf{w}_u + \mathbf{w}_u^T \mathbf{M}_{uk} \mathbf{w}_k + \mathbf{w}_u^T \mathbf{M}_{uu} \mathbf{w}_u. \tag{3.28}$$

Its minimum with respect to $\mathbf{w}_u$ is given by

$$\mathbf{0} \stackrel{!}{=} \frac{\partial R(\mathbf{w})}{\partial \mathbf{w}_u} = 2\mathbf{M}_{uk} \mathbf{w}_k + 2\mathbf{M}_{uu} \mathbf{w}_u \tag{3.29}$$

$$\mathbf{w}_u = -\left(\mathbf{M}_{uu}\right)^{-1} \mathbf{M}_{uk} \mathbf{w}_k \tag{3.30}$$

Figure 3.6 illustrates this method on a 1-D PSOM$^+$ with 8 nodes.

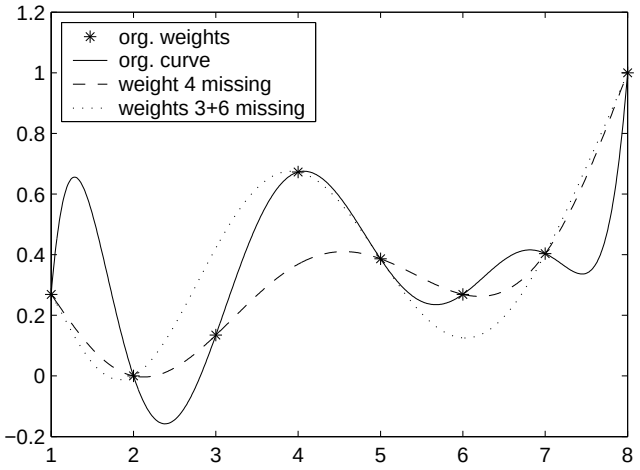


Figure 3.6: PSOM$^+$ mappings from incomplete grid-organized data.

3.3.4. Per-weight smoothing

If the scalar smoothing parameter λ in (3.27) is replaced by a diagonal matrix $\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \dots)$ of per-weight smoothing parameters, the two problems described in the last sections can be merged into one. We minimize a term corresponding to (3.24) divided by λ,

$$E(\mathbf{w}, \mathbf{\Lambda}) = \sum_{i=1}^{N} \frac{1}{\lambda_i} (w_i - \tilde{w}_i)^2 + R(\mathbf{w}) \tag{3.31}$$

$$= (\mathbf{w} - \tilde{\mathbf{w}})^T \mathbf{\Lambda}^{-1} (\mathbf{w} - \tilde{\mathbf{w}}) + \mathbf{w}^T \mathbf{M} \mathbf{w}, \tag{3.32}$$

which yields optimal weights

$$\mathbf{w}_{opt} = (\mathbf{I} + \mathbf{\Lambda}\mathbf{M})^{-1} \tilde{\mathbf{w}}. \tag{3.33}$$

All exactly known weights can be assigned $\lambda = 0$, whereas missing weights would be interpreted as known, but infinitely noisy and therefore be endowed with a high value of λ. Intermediate values can be used if information about the noise level at a particular grid position is available, which reveals the close relation of this PSOM variant to general heteroscedastic regression (see, e.g., Yuan and Wahba, 2004).

Figure 3.7 shows an example using the same toy data as before. The weights 3 and 6 are treated as unknown or noisy and assigned a non-zero λ. All other weights are fixed ($\lambda = 0$). Note the "morphing" between the curves for the totally known and totally unknown case.

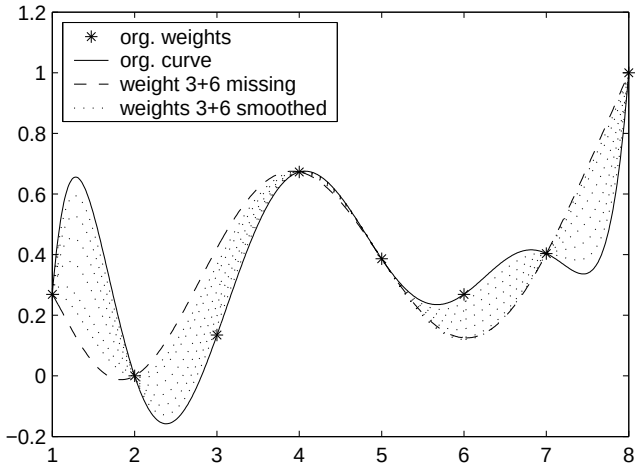


Figure 3.7: PSOM$^+$ mappings from incomplete or partially noisy grid-organized data.

3.3.5. Non-grid-organized data

In this section, we describe how to build a PSOM mapping if the training data is not ordered on a grid, but rather is given as a set of M input-output pairs[6] $(\mathbf{x}_i, y_i)$. In order to construct a PSOM$^+$ mapping from such data, we propose the following: at first, specify a grid that spans a hyper-rectangle just large enough to embed the input data $\mathbf{x}_i$. If you have no idea how complex the mapping is, use as many nodes as is computationally feasible. The spacing of the nodes along the different axes is arbitrary, because it will change only the representation of the multi-dimensional polynomial that is fitted to the data, but not its shape.

Then, construct the smoothest mapping passing through the training data, that is

$$\text{minimize} \quad E(\mathbf{w}) \quad \text{subject to} \quad y_i \stackrel{!}{=} w(\mathbf{s}_i) = \sum_{\mathbf{g}} w_{\mathbf{g}} b_{\mathbf{g}}(\mathbf{s}_i) \quad i = 1 \ldots M. \tag{3.34}$$

By defining a matrix $\mathbf{B}$ with components $b_{ig} = b_g(\mathbf{s}_i)$, the constraints can be written as $\mathbf{y} = \mathbf{B}\mathbf{w}$. The optimization problem (3.34) only features linear equality constraints and thus can be solved by null-space methods. In particular, we first determine a feasible

[6] Again, we can treat multiple output data dimensions separately.

solution $\mathbf{y} = \mathbf{B}\mathbf{w}_0$ and also the matrix $\mathbf{Z} \in \mathbb{R}^{N \times K}$ that spans the null-space of $\mathbf{B}$, that is, the columns $\mathbf{z}_k$ of $\mathbf{Z}$ are orthogonal and satisfy $\mathbf{B}\mathbf{z}_k = \mathbf{0}$. Thus, for any vector $\mathbf{v} \in \mathbb{R}^K$, the weight vector $\mathbf{w} = \mathbf{w}_0 + \mathbf{Z}\mathbf{v}$ also satisfies the constraints. In order to find the weights of the smoothest mapping, we minimize

$$\begin{aligned} R(\mathbf{v}) &= (\mathbf{w}_0 + \mathbf{Z}\mathbf{v})^T \mathbf{M} (\mathbf{w}_0 + \mathbf{Z}\mathbf{v}) && (3.35) \\ &= \mathbf{w}_0^T \mathbf{M} \mathbf{w}_0 + 2\mathbf{v}^T \mathbf{Z}^T \mathbf{M} \mathbf{w}_0 + \mathbf{v}^T \mathbf{Z}^T \mathbf{M} \mathbf{Z} \mathbf{v} && (3.36) \end{aligned}$$

with respect to $\mathbf{v}$, which yields

$$\mathbf{v}_{opt} = \left(\mathbf{Z}^T \mathbf{M} \mathbf{Z}\right)^{-1} \mathbf{Z}^T \mathbf{M} \mathbf{w}_0. \tag{3.37}$$

If the constraints are infeasible, i.e. there are not enough nodes to interpolate the data by a multi-dimensional polynomial, one can use the pseudo-inverse of $\mathbf{B}$ to get an approximate solution $\mathbf{w}_0$.

In case the output data is noisy, it makes no sense to use the hard constraints of (3.34). Instead we proceed similar to section 3.3.2 and minimize a weighted sum of the smoothness measure and the distance between observed and reconstructed data:

$$\begin{aligned} \mathbf{w}_{opt} &= \arg\min_{\mathbf{w}} \left[\|\mathbf{y} - \mathbf{B}\mathbf{w}\|^2 + \lambda R(\mathbf{w}) \right] && (3.38) \\ &= \arg\min_{\mathbf{w}} \left[\mathbf{y} - \mathbf{B}\mathbf{w})^T (\mathbf{y} - \mathbf{B}\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{M} \mathbf{w} \right] && (3.39) \\ &= (\mathbf{B}^T \mathbf{B} + \lambda \mathbf{M})^{-1} \mathbf{B}^T \mathbf{y}. && (3.40) \end{aligned}$$

In this case, the PSOM$^+$ algorithm effectively turns into a basis function approach to penalized least squares regression, and indeed (3.40) has the same form as the general solution to this problem (cf. e.g. Green and Silverman, 1994, p.46).

As an illustration, Fig. 3.8 shows the resulting mappings of a 1-D PSOM$^+$ with 8 nodes placed at $a_i = i = 1 \ldots 8$. The task was to interpolate 6 training data samples either (i) exactly or (ii) approximately by specification of a smoothing factor.

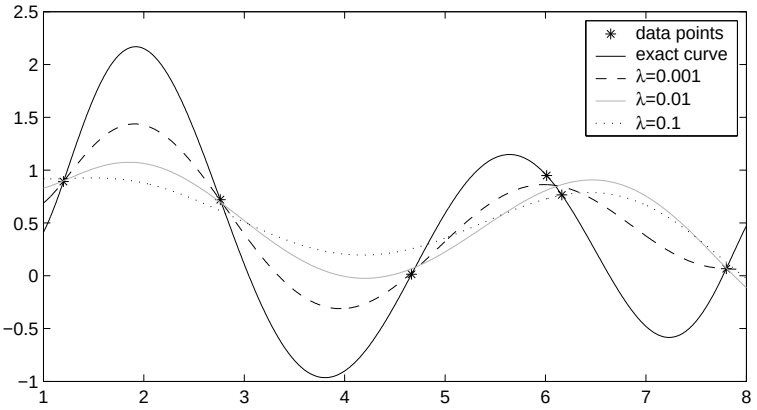


Figure 3.8: PSOM$^+$ from non-grid-organized toy data.

3.4. PSOM$^+$ model of PA-10 kinematics

In this section, we demonstrate our PSOM$^+$ extensions on the kinematics learning task already described in section 3.2. Again, we simulated learning the 3D wrist position as a function of the first four joint angles, and based our experiments on an 8x8x8x8 PSOM with Chebyshev spacing. Each grid axis represents one joint and spans its respective range, so the manifold parameter $\mathbf{x}$ directly corresponds to the four joint angles. As a starting point, we analytically calculated the "true" weights (wrist positions) for all nodes (joint angle sets). Furthermore, we generated 1000 random postures ($\mathbf{x}$) on which we compared the analytic forward kinematics to the result of the PSOM$^+$ mapping.

In our first experiment, we added Gaussian noise of standard deviation $\sigma = 10$mm to all (analytically computed) weights. Then, we de-noised the PSOM$^+$ mapping with different values for λ (cf. section 3.3.2). Figure 3.9 shows the positional error on the test set, as well as the mean deviation between the original noisy and the de-noised weights. Note that a slight denoising (e.g. $\lambda = 0.002$) achieves a lower mean positional error, but that the decrease lies within the standard deviation.

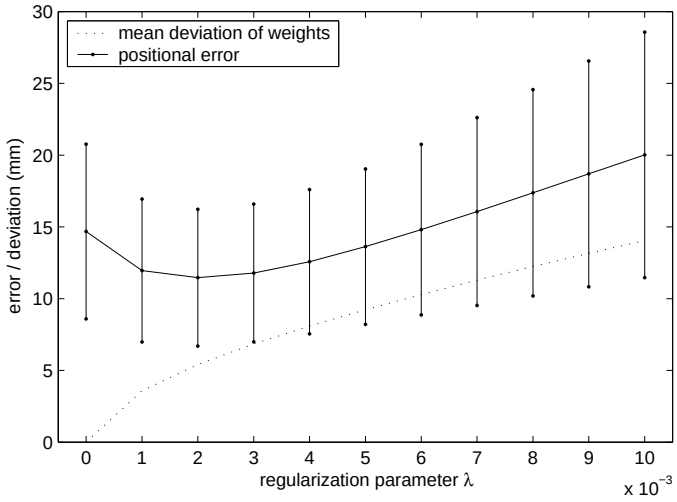


Figure 3.9: PA10 wrist kinematics from noisy data. The position $\lambda = 0$ corresponds to the PSOM mapping without de-noising. The positional error is depicted by bars indicating the mean $\pm$ one standard deviation.

As a second experiment, we randomly selected $N_{miss} = 40$ (100, 400) nodes and treated their weights as missing. In addition to calculating optimal weights for these nodes in the sense of section 3.3.3, we alternatively simply averaged (i) the neighboring weights and (ii) all "known" weights to fill the gaps. Table 3.2 shows the resulting mean positional error on the set of 1000 test postures. As can be seen, our procedure outperforms the two "naive" approaches by more than a magnitude regarding the accuracy.

In our third experiment, we randomly generated $M = 2048$ (3072, 4096, 6144) joint postures as well as their corresponding analytically determined wrist positions. Then, we constructed a PSOM$^+$ mapping only from these (non-grid organized) data samples, using the method of section 3.3.5. Table 2 depicts the mean positional error on the

	positional error in mm: mean ± std. dev.		
N_{miss}	**optimized**	local average	global average
40	**0.64 ± 0.45**	7.53 ± 15.38	15.65 ± 30.53
100	**1.03 ± 1.18**	25.86 ± 40.21	56.29 ± 85.41
400	**2.12 ± 2.21**	62.90 ± 60.67	140.32 ± 136.02

Table 3.2: PA10 wrist kinematics from missing data. As a comparison, the mean (± std. dev.) of the positional error for the complete PSOM mapping (cf. Table 3.1) is 0.54 ± 0.27.

test set. Note that for 4096 random training samples, the resulting mean error is much higher as compared to the PSOM mappings that were built from a grid-ordered training set of the same size. Even utilizing 6144 randomly sampled training samples did not yield the same level of accuracy. Still, if grid-ordered training data is not available, the PSOM$^+$ allows us to learn a mapping of reasonable accuracy.

Size of training set	positional error in mm mean (std dev)
2048	7.45 ± 15.44
3072	4.80 ± 13.21
4096	2.85 ± 7.96
6144	1.55 ± 4.75

Table 3.3: PSOM$^+$ model of the PA10 wrist kinematics from non-grid-organized training data.

3.5. Unsupervised learning of manifolds with the PSOM$^+$

Up to now, the PSOM algorithm has only been applied in a basically supervised setting. Either the PSOM is built upon a readily trained SOM, or it is created from an otherwise topologically ordered training set. The inclusion of the smoothness measure as a regularization term and the ability to handle non-grid-organized data, however, allow us to derive an unsupervised manifold learning algorithm which is similar in spirit to the Regularized Principal Manifolds algorithm from section 2.6.2. We use an objective function matching (2.67), that is, a weighted sum of the mean reconstruction error of M data vectors $\mathbf{y}_i \in \mathbb{R}^d$ and the regularization term (3.22):

$$E(\mathbf{W}, \lambda) = \frac{1}{M} \sum_{i=1}^{M} \min_{\mathbf{x}_i} \|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i; \mathbf{W}))\|^2 + \lambda R(\mathbf{W}), \tag{3.41}$$

where we chose the notation $\mathbf{f}(\mathbf{x}; \mathbf{W})$ to indicate that the weights $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2 \ldots \mathbf{w}_d)$ alone determine the shape of the mapping. Like Smola et al. (2001) specify their sample points $\mathbf{r}_i$ in advance, we do so with the underlying grid.

As the main differences, our function model is a multi-dimensional polynomial instead of a kernel expansion, and our regularization term is the smoothness measure based on second derivatives (3.15) instead of a regularizer implied by the kernel.

In conformity with the proposition of Smola et al. (2001), we also initialize the PSOM$^+$ model by PCA, and then fit the model by alternating projection and adaption steps. If $\mathbf{V}_q$ denotes the matrix of eigenvectors corresponding to the q largest eigenvalues of the sample covariance matrix of the data $\{\mathbf{y}_i\}$ (cf. Sec. 2.2), then we set the initial weights of the PSOM$^+$ to

$$\mathbf{w_g} = \bar{\mathbf{y}} + \mathbf{V}_q \mathbf{S} \mathbf{a_g}, \tag{3.42}$$

where $\bar{\mathbf{y}}$ is the mean of the data and $\mathbf{S} = \mathrm{diag}(s_1, \ldots, s_q)$ is a diagonal scaling matrix. We adjust s_i in a way that all orthogonal projections onto the PCA manifold are covered by the range of the PSOM mapping, that is, $s_i = \max_j |\mathbf{v}_i^T(\mathbf{y}_j - \bar{\mathbf{y}})|$, where $i = 1 \ldots q,\ j = 1 \ldots M$ and we assume that the domain of the grid coordinates is $[-1; 1]^q$.

In the projection step, we solve the bound-constrained optimization problem

$$\mathbf{x}_i = \arg \min_{\mathbf{x} \in [-1;1]^q} \|\mathbf{y}_i - \mathbf{f}(\mathbf{x}; \mathbf{W})\|^2 \qquad \forall i = 1 \ldots M, \tag{3.43}$$

that is, for every data vector $\mathbf{y}_i$ we seek the optimal position $\mathbf{x}_i$ within the domain of the grid. This can be efficiently achieved by the Levenberg-Marquardt algorithm, which has also been applied to the PSOM best match search by Walter and Ritter (1996). The optimization procedure can start from the best matching node, or from the solution $\mathbf{x}_i$ of a preceding projection step. In the adaption step, we fix $\{\mathbf{x}_i\}$ and calculate optimal weights as in (3.40). In general, both steps will decrease (or keep constant) the regularized risk (3.41), which is immediately clear for the adaption step. During projection, the roughness measure of the manifold does not change, and by definition the best match search decreases the reconstruction error – it might, however, get stuck in a bad local minimum.

As an example of this approach, we fitted a 12x12 Chebyshev PSOM$^+$ model to an artificial "fish bowl" dataset. The "fish bowl" is a 2-D manifold embedded in $\mathbb{R}^2$ and can be described as the surface of a sphere with a clipped-off south pole[7]. To generate the data, we calculated 1000 samples

$$\mathbf{y} = r \begin{pmatrix} \sqrt{1-c^2}\cos(\phi) \\ \sqrt{1-c^2}\sin(\phi) \\ c \end{pmatrix} + \mathbf{u} \tag{3.44}$$

with a radius $r = 10$, $c = \cos(\theta)$ uniformly distributed in $[-0.7; 1]$, and ϕ uniformly distributed in $[0; 2\pi]$. $\mathbf{u}$ denotes isotropic Gaussian noise $\mathcal{N}(0, \sigma^2)$, where we set $\sigma = 0.1$.

We initialized the PSOM$^+$ via PCA as described above, and then carried out 30 projection/adaption steps, where we decreased the regularization parameter λ by the schedule $\lambda_k = 10 \cdot 0.8^k$ with $k = 1, 2, \ldots 30$ denoting the iteration counter. The initial model, some intermediate steps, and the final PSOM$^+$ model are shown in Fig. 3.10. Please note how decreasing λ results in a slow, controlled adaption of the PSOM$^+$ manifold to the data, comparable to the deterministic annealing scheme that Rose et al. (1990) utilized for clustering. Indeed, with that scheme a much better solution is found than for example by carrying out 30 iterations with λ fixed at the final value $\lambda = 10 * 0.8^{30} \approx 0.0124$, which is illustrated in Fig. 3.11. The annealed PSOM$^+$ model is smoother and also features a smaller mean reconstruction error (0.0386 vs. 0.0764).

[7] Thus, actually the "fish bowl" lies on its top.

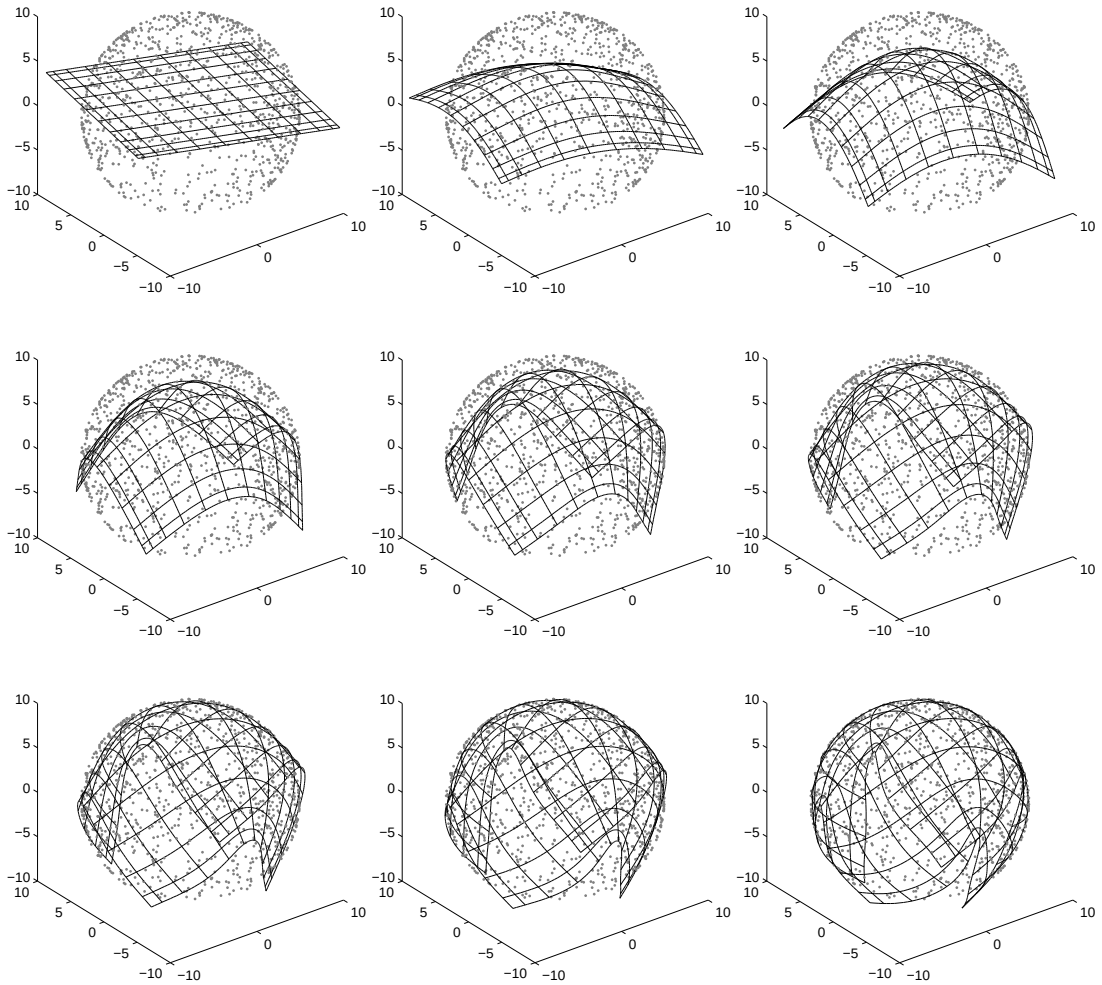


Figure 3.10: PSOM$^+$ model of the "fish bowl". The top left plot and the bottom right plot depict the PCA initialization and the final model ($k = 30$), respectively. The remaining plots show intermediate results for $k = 1, 2, 4, 7, 10, 15$, and 20.

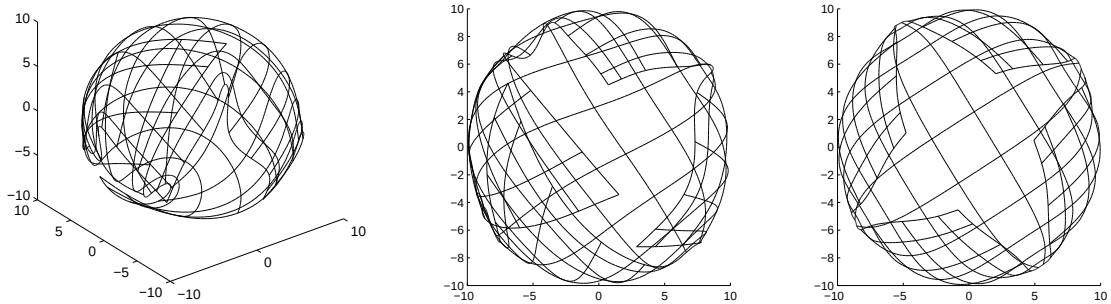


Figure 3.11: PSOM$^+$ model of the "fish bowl". Left: resulting model without annealing. Middle: the same model viewed from below. Right: The annealed model from Fig. 3.10 viewed from below.

3.6. Discussion

The PSOM algorithm has often been applied in learning tasks in robotics, where highly accurate approximation models can be built if only the training data has the right structure. Especially in kinematics learning tasks, the function model of the PSOM seems very appropriate, because e.g. the end-effector position typically varies nonlinearly but smoothly as a function of the joint angles. Multiple joints interact strongly, which is accounted for by the PSOM's products of 1-D polynomials. A particularly appealing feature of the PSOM is the ability to represent the forward and inverse kinematics in one united manifold.

In this chapter of the thesis, we presented an approach to regularize PSOM mappings based on minimizing their overall roughness. Our method allows us to construct PSOMs from noisy and not necessarily grid-organized, or incomplete training data. As an application, we demonstrated the approach for learning a part of the PA-10 kinematics, where in particular the "missing data" extension yielded promising results.

We also indicated how the extended PSOM$^+$ model can be utilized as an unsupervised manifold learning method, and presented an algorithm similar to the Regularized Principal Manifolds (cf. Sec. 2.6.2). In contrast to Smola et al. (2001), we did not provide an analysis of convergence bounds, but only presented an example experiment as a proof of concept.

Indeed, we still see the PSOM's scope of application mainly in robotics: the latent space of the PSOM$^+$ is still a hyper-rectangle, and the function model is still a global multi-dimensional polynomial of possibly high degree. Both properties do not necessarily match a general data distribution well. The "fish bowl", as depicted in Figs. 3.10 and 3.11, demonstrates this drawback by the sharp bends of the PSOM$^+$ manifold's borders, which here implies a distortion of the topology.

We did not provide any guidance for the selection of the grid size and the regularization parameter λ. For the corresponding problem of selecting support points $\mathbf{r}_i$ in section 2.6.2, Smola et al. (2001, p. 184) propose to take "as many as one may afford in terms of computational cost." This strategy can also be applied for the PSOM$^+$, provided the regularization parameter λ (which Smola et al. chose subjectively in their experiments) is large enough to prevent heavy oscillations of the polynomials. A principled adjustment of λ is possible by utilizing hold-out data, cross-validation, or similar techniques.

4. Unsupervised Kernel Regression

Unsupervised Kernel Regression (UKR) is a non-parametric approach for learning of principal manifolds. It has been introduced as an unsupervised counterpart of the Nadaraya-Watson kernel regression estimator by Meinicke et al. (2005). In essence, UKR uses the Nadaraya-Watson estimator (please see the next section) to find both a latent space representation of a dataset and a smooth mapping from latent space back to the space of the original data.

Through this, UKR has two main advantages over other dimension reduction and manifold learning algorithms: firstly, one can apply leave-one-out cross-validation (LOO-CV) as an automatic complexity control without additional computational cost. Secondly, defining a complete UKR model requires the a priori specification of only very few parameters. In fact, if LOO-CV is used for regularization, the only choices to be made are the desired dimensionality of latent space and the shape (but not the bandwidth) of a density kernel.

These properties distinguish UKR from many of the algorithms that were presented in the preceding chapters. When utilizing auto-associative neural networks (Sec. 2.3), a certain network structure has to be set up: The bottleneck layer is sized to the desired dimensionality, while the number of neurons in further hidden layers controls the complexity or capacity of the mapping. The Self-Organizing Map (Sec. 2.4) requires the specification of a grid of neurons and two time-varying parameters that control the weight adaption process. Principal curve algorithms also rely on certain adjustments, i.e. the pre-factor of a roughness penalty (Sec. 2.5.1), a heuristical threshold to decide about adding new vertices (Sec. 2.5.2), or a smoothing parameter σ (Sec. 2.5.3).

The Generative Topographic Mapping (Sec. 2.6.1), modelled after the SOM, is also based on a grid in latent space and additionally requires the specification of parameters for the basis functions. An approach to automatically infer some of these parameters has been described by Bishop et al. (1998a), but in their experiments the number of basis functions and the size of the grid was chosen ad hoc. Similarly, Regularized Principal Manifolds (Sec. 2.6.2) require a pre-specified collection of support points as well as the choice of a kernel. In their experiments, Smola et al. (2001) did not motivate their selection of the width of the Gaussian kernel or the choice of the number of support points, and they adjusted the regularization parameter by hand.

Furthermore, we listed several methods for nonlinear dimension reduction which do not yield an explicit manifold model (or a discrete approximation in case of the SOM), but only a lower dimensional representation of the data. Among methods that iteratively seek an optimum of sometimes highly non-convex objective functions are the Curvilinear Component Analysis (CCA, Sec. 2.7.3), which is implemented using time-varying neighborhood and learning rate parameters. Curvilinear Distance Analysis (Sec. 2.7.4) has been proposed as an improved version of CCA featuring some automatic parameter selection. Nonlinear spectral embedding methods (Sec. 2.8) feature a convex objec-

tive function, and the optimum can be found efficiently by solving an eigenproblem. However, each of these methods need the specification of a (sometimes very critical) parameter that describes the size of neighborhoods in data space.

This chapter of the thesis is devoted to a thorough description of the UKR algorithm as originally published by Meinicke et al. (2005). In order to lay the grounds, we will first study the Nadaraya-Watson estimator.

4.1. The Nadaraya-Watson estimator

For now, we return to the problem of approximating the functional relationship between two random variables x and y, which for simplicity we assume to be univariate. First, we recall the definition of the regression function (1.5) and rewrite it in terms of the joint and marginal distributions $p(x, y)$ and $p(x)$:

$$f^*(x) = \langle y|x \rangle = \int y\, p(y|x)\, dy = \int y\, \frac{p(x,y)}{p(x)}\, dy. \tag{4.1}$$

If our goal is to estimate $f^*(x)$ from samples $\{(x_i, y_i) \mid i = 1 \dots N\}$, we could utilize a kernel density estimate of the joint distribution, given by

$$\hat{p}(x,y) = \frac{1}{N} \sum_{i=1}^{N} K_{h_x}(x - x_i) K_{h_y}(y - y_i), \tag{4.2}$$

where $K_h(\cdot)$ is a density kernel function with a *bandwidth* parameter h and the properties

$$\int K_h(z)dz = 1 \quad \text{and} \quad \int z K_h(z)dz = 0. \tag{4.3}$$

Note that the bandwidth parameters h_x and h_y need not be equal. An example of a density kernel is the Gaussian kernel

$$K_h(z) = \frac{1}{h} K\left(\frac{z}{h}\right) = \frac{1}{\sqrt{2\pi}h} \exp\left(-\frac{1}{2}\frac{z^2}{h^2}\right). \tag{4.4}$$

Please note that while the Gaussian kernel is also a Mercer kernel (cf. Sec. 1.5.4), in general one has to distinguish between the two classes of kernels. In this thesis, a density kernel is denoted by an upper-case letter and a single argument $K(\cdot)$, while Mercer kernels are denoted by lower-case letters and two arguments $k(\cdot, \cdot)$.

In the limit of infinitely small bandwidths ($h \to 0$), the kernel estimate of the joint distribution becomes equal to the empirical data distribution

$$p_{emp}(x,y) = \frac{1}{N} \sum_{i=1}^{N} \delta(x - x_i)\delta(y - y_i), \tag{4.5}$$

while larger bandwidths "smear" that peaky distribution. An estimate $\hat{p}(x)$ of the marginal density can be retrieved by integration over y:

$$\begin{aligned} \hat{p}(x) &= \int \hat{p}(x,y)\, dy = \int \frac{1}{N} \sum_{i=1}^{N} K_{h_x}(x - x_i) K_{h_y}(y - y_i) dy \\ &= \frac{1}{N} \sum_{i=1}^{N} K_{h_x}(x - x_i) \int K_{h_y}(y - y_i) dy. \end{aligned} \tag{4.6}$$

Since according to (4.3) the integral over a density kernel is 1, we retrieve a kernel density estimate of x alone

$$\hat{p}(x) = \frac{1}{N} \sum_{i=1}^{N} K_{h_x}(x - x_i), \tag{4.7}$$

which is known as the Rosenblatt-Parzen estimator (Parzen, 1962). Insertion of (4.7) and (4.2) into (4.1) yields

$$\begin{aligned} \hat{f}(x) &= \int y \frac{\frac{1}{N} \sum_i K_{h_x}(x - x_i) K_{h_y}(y - y_i)}{\frac{1}{N} \sum_j K_{h_x}(x - x_j)} dy \\ &= \sum_i \frac{K_{h_x}(x - x_i)}{\sum_j K_{h_x}(x - x_j)} \int y K_{h_y}(y - y_i) dy, \end{aligned} \tag{4.8}$$

where the last integral can be solved by substituting $y = z + y_i$ and taking note of the conditions (4.3). Finally, the resulting estimator (Nadaraya, 1964; Watson, 1964) is given by

$$\hat{f}(x; h_x) = \sum_i y_i \frac{K_{h_x}(x - x_i)}{\sum_j K_{h_x}(x - x_j)}. \tag{4.9}$$

Please note that the bandwidth h_y does not play a role anymore. The only parameters of this estimator are the choice of the density kernel, which is of relatively low importance (Härdle and Marron, 1985), and the bandwidth $h = h_x$, which is crucial.

The Nadaraya-Watson estimator (4.9) performs a local averaging of the output data, where the kernel functions $K_h(x - x_i)$ determine how strongly y_i contributes to the function value when evaluated at x. The bandwidth controls the span of the local averaging, and therefore the smoothness of the mapping: In the limit of an infinitely large bandwidth, the ratio in (4.9) becomes $\frac{1}{N}$ at any point x, and thus the estimator constantly yields the mean of the data. In the opposite limit, when $K_h(x - x_i) \to \delta(x - x_i)$, the estimator yields a pointwise constant mapping with $\hat{f}(x_i) = y_i$, that is the samples are interpolated. An illustration of this dependency on the bandwidth (or smoothing parameter) h is provided in Fig. 4.1.

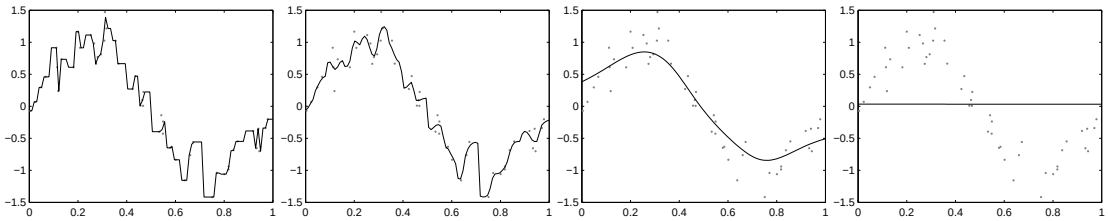


Figure 4.1: Results of using the Nadaraya-Watson estimator with different bandwidth parameters. The data are 50 noisy samples from a sine curve. From left to right: $h = 0.001$, $h = 0.01$, $h = 0.1$, and $h = 10$.

4.1.1. Choice of smoothing parameter

A particularly simple and appealing choice of the parameter h is given by leave-one-out cross-validation, which was in this context introduced by Clark (1975). The idea is to

select the bandwidth h_{cv} that minimizes the quantity

$$CV(h) = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \hat{f}_{-i}(x_i;h)\right)^2, \tag{4.10}$$

where $\hat{f}_{-i}(\cdot)$ denotes the Nadaraya-Watson estimator for a dataset with the pair (x_i, y_i) left out:

$$\hat{f}_{-i}(x;h) = \sum_{k\neq i} y_k \frac{K_{h_x}(x-x_k)}{\sum_{j\neq i} K_{h_x}(x-x_j)}. \tag{4.11}$$

Härdle and Marron (1985) showed that under mild assumptions this choice is asymptotically optimal, that is, in the limit $N \to \infty$ the bandwidth h_{cv} yields the best estimate with respect to several criteria like the integrated squared deviation between $\hat{f}(\cdot)$ and the true regression function[1].

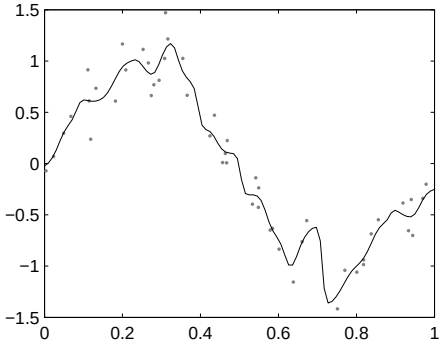


Figure 4.2: Results of using the Nadaraya-Watson estimator with the optimal bandwidth $h_{cv} = 0.0175$ as determined by leave-one-out cross-validation. The curve does not match the underlying sinus function well, but keep in mind that the sample size is only $N = 50$.

4.1.2. Multivariate generalization and further kernels

The Nadaraya-Watson estimator can easily be generalized to the case of multivariate input and output data, yielding

$$\mathbf{f}(\mathbf{x};\mathbf{H}) = \sum_i \mathbf{y}_i \frac{K_{\mathbf{H}}(\mathbf{x}-\mathbf{x}_i)}{\sum_j K_{\mathbf{H}}(\mathbf{x}-\mathbf{x}_j)}. \tag{4.12}$$

Here, the one-dimensional smoothing parameter h is replaced by a diagonal matrix $\mathbf{H} = \mathrm{diag}(h_1, h_2, \ldots, h_q)$, and for example the multivariate Gaussian kernel is given by

$$K_{\mathbf{H}}(\mathbf{x}) = \frac{1}{\det(\mathbf{H})} K\left(\mathbf{H}^{-1}\mathbf{x}\right) = \frac{1}{(2\pi)^{\frac{q}{2}}\det(\mathbf{H})} \exp\left(-\frac{1}{2}\left\|\mathbf{H}^{-1}\mathbf{x}\right\|^2\right). \tag{4.13}$$

[1] More precisely, Härdle and Marron (1985) included a weighting factor $w(x_i)$ within the CV-error (4.10), which e.g. accounts for restricting the domain where the optimality criteria should hold. We can think of $w(x_i)$ as being constant within a finite region of interest, and zero elsewhere.

Other popular density kernels include the Epanechnikov kernel ($h = 1$)

$$K(x) = \frac{3}{4}\left[1 - x^2\right]_+ = \begin{cases} \frac{3}{4}\left(1 - x^2\right) & |x| < 1 \\ 0 & |x| \geq 1, \end{cases} \tag{4.14}$$

which is continuous, but not differentiable at $|x| = 1$, and the Quartic kernel ($h = 1$)

$$K(x) = \frac{15}{16}\left[1 - x^2\right]_+^2 = \begin{cases} \frac{15}{16}\left(1 - x^2\right)^2 & |x| < 1 \\ 0 & |x| \geq 1, \end{cases} \tag{4.15}$$

which does not share this problem. Both kernels have finite support and therefore lead to an improved computational efficiency: when evaluating the Nadaraya-Watson estimator at a point x_0, typically only a few terms $K(x_0 - x_i)$ will be non-zero, and so the linear combination of output data samples y_i will only involve few components.

As a multivariate generalization of these kernels, one can use a product of 1-D kernels

$$K(\mathbf{x}) \propto \prod_{i=1}^{q} K(x_i) \tag{4.16}$$

or a spherically symmetric kernel

$$K(\mathbf{x}) \propto K(\|\mathbf{x}\|). \tag{4.17}$$

In case of the Gaussian kernel, both variants are the same. Here, we overloaded the notation $K(\cdot)$ and also left out the normalization pre-factor. Please note that it is not necessary to include the latter, because any pre-factor cancels out in the ratios of (4.9) and (4.12), respectively.

4.2. Derivation of UKR

As stated in section 1.4.1, the goal of manifold learning is to find both a faithful lower dimensional representation (latent variables) $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) \in \mathbb{R}^{q \times N}$ of an observed dataset $\mathbf{Y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N) \in \mathbb{R}^{d \times N}$ and a corresponding functional relationship. Inspired by the generalized regression framework of Meinicke (2000), the basic idea of UKR is to address this problem by using the Nadaraya-Watson estimator as the mapping from latent space to data space, whereby the latent variables $\mathbf{X}$ take the role of the input data and are treated as *parameters* of the regression function.

As a first consequence, the bandwidth parameter $\mathbf{H}$ of (4.12) can be dropped, or more precisely we can use a unit bandwidth, because any bandwidth $\neq 1$ can be accounted for by a corresponding re-scaling $\mathbf{X} \to \mathbf{H}^{-1}\mathbf{X}$, and the scaling of the parameters $\mathbf{X}$ is free. Accordingly, we will write the UKR regression function as

$$\mathbf{f}(\mathbf{x}; \mathbf{X}) = \sum_{i=1}^{N} \mathbf{y}_i \frac{K(\mathbf{x} - \mathbf{x}_i)}{\sum_j K(\mathbf{x} - \mathbf{x}_j)} = \sum_{i=1}^{N} \mathbf{y}_i b_i(\mathbf{x}; \mathbf{X}) = \mathbf{Y}\mathbf{b}(\mathbf{x}; \mathbf{X}), \tag{4.18}$$

where for convenience we introduced a vector $\mathbf{b}(\cdot) \in \mathbb{R}^N$ of basis functions containing the ratio of the kernels. At first sight, the notation $\mathbf{f}(\mathbf{x}; \mathbf{X})$ may seem a little awkward, so for clarification we repeat that $\mathbf{x}$ is just the location where we want to evaluate the

function, while $\mathbf{X}$ are the low-dimensional representations of the data $\mathbf{Y}$ that the UKR model is fitted to, and at the same time the parameters of the function model. In principle, the function value also depends on $\mathbf{Y}$, but we leave it out from the parameter list, since $\mathbf{Y}$ is fixed and not adapted during training of the model.

While the form "matrix × vector of basis functions" of (4.18) is already familiar from e.g. the GTM (Sec. 2.6.1), the function model of UKR is quite different from the traditional basis function approach. Whereas the GTM has fixed (pre-specified) basis functions and an adaptable parameter matrix, in UKR the matrix is fixed, and the flexibility lies in the basis functions, which in turn need no specification beyond the choice of a density kernel. Just as in supervised kernel regression, the latter is qualitatively rather unimportant, but it does have an impact on the computational efficiency of the algorithm. Finite support kernels lead to sparse vectors of basis functions and therefore allow a quicker evaluation of the regression function.

In order to derive an alternative notation, please note that the denominator in (4.18) is proportional to the Rosenblatt-Parzen estimator of the latent space density

$$p(\mathbf{x};\mathbf{X}) = \frac{1}{N}\sum_{j=1} K(\mathbf{x}-\mathbf{x}_j). \tag{4.19}$$

Therefore, we could also express the basis functions by

$$b_i(\mathbf{x};\mathbf{X}) = \frac{K(\mathbf{x}-\mathbf{x}_i)}{Np(\mathbf{x};\mathbf{X})}. \tag{4.20}$$

For the sake of convenience, we will sometimes drop the extra argument $\mathbf{X}$ in the following.

4.2.1. UKR manifold and generalization

We define the UKR manifold as the range of $\mathbf{f}(\mathbf{x};\mathbf{X})$, where the domain is restricted to some finite subset $\mathcal{X} \subset \mathbb{R}^q$, that is

$$\mathcal{M} = \{\mathbf{y} = \mathbf{f}(\mathbf{x};\mathbf{X}) \mid \mathbf{x} \in \mathcal{X}\}. \tag{4.21}$$

At any point $\mathbf{x}$ where $\mathbf{f}(\mathbf{x};\mathbf{X}) = \mathbf{Y}\mathbf{b}(\mathbf{x};\mathbf{X})$ is evaluated, the elements $b_i(\mathbf{x};\mathbf{X})$ sum to 1, so the function value is a convex combination of data vectors. Nearby latent variables $\mathbf{x}_i$ yield larger values $K(\mathbf{x}-\mathbf{x}_i)$ and thus larger weights within that combination. Therefore, if nearby latent variables belong to data vectors that are also nearby, the UKR manifold should fit the data closely and smoothly. Can we somehow enforce such a reasonable shape of the manifold?

With $\mathbf{f}(\mathbf{x};\mathbf{X})$ being a convex combination, it is clear that the manifold $\mathcal{M}$ must be a subset of the span of the data $\mathbf{Y}$. Unfortunately this does not imply proper generalization, since even within that span the manifold may exhibit an arbitrarily wriggly behavior. Analogously to the length constraint of the principal curves of Kégl et al. (2000), we have to make sure that the spatial extension of the UKR manifold is bounded.

For simplicity, we first restrict our analysis to the case $q = 1$, that is, the UKR manifold being a curve. The length of that curve is given by

$$\begin{aligned} L &= \int_{\mathcal{X}} \|\mathbf{f}'(x)\| dx = \int_{\mathcal{X}} \|\mathbf{Y}\mathbf{b}'(x)\| dx \leq \|\mathbf{Y}\|_2 \int_{\mathcal{X}} \|\mathbf{b}'(x)\| dx \\ &\leq \|\mathbf{Y}\|_2 \max_{x \in \mathcal{X}} \|\mathbf{b}'(x)\| \int_{\mathcal{X}} dx, \end{aligned} \tag{4.22}$$

where $\mathbf{b}'(x)$ denotes the vector of all derivatives at x with components

$$b_i'(x) = \frac{K'(x - x_i)}{Np(x)} - \frac{K(x - x_i)p'(x)}{Np^2(x)}. \tag{4.23}$$

Thus, for a finite domain $\mathcal{X}$ and a continuously differentiable kernel $K(\cdot)$, the length of the curve is upper bounded if the density $p(x)$ is lower bounded by some positive non-zero constant.

To generalize our argument to the case $q > 1$, we have to show that the manifold's area or volume is bounded. For this end, please recall that an infinitesimal volume element $dx_1 dx_2 \dots dx_q$ at a point $\mathbf{x}$ in latent space transforms into a hyper-parallelepiped at $\mathbf{f}(\mathbf{x}; \mathbf{X})$ in data space, spanned by the vectors $\mathbf{J}_1 dx_1, \mathbf{J}_2 dx_2 \dots \mathbf{J}_q dx_q$. Here, $\mathbf{J}_i$ denotes the i-th column of the Jacobian of $\mathbf{f}(\mathbf{x}; \mathbf{X})$. Since the volume of a hyper-parallelepiped is less or equal to the volume of a hyper-rectangle of corresponding side lengths, and since the norm of each $\mathbf{J}_i$ is less or equal to the largest singular value (and thus, the matrix norm) of the complete Jacobian $\mathbf{J}_\mathbf{f}$, the volume of our infinitesimal hyper-parallelepiped is bounded by $\|\mathbf{J}_\mathbf{f}\|_2^q dx_1 dx_2 \dots dx_q$.

We can use this inequality to estimate the complete manifold's volume V by integration over the latent domain:

$$\begin{aligned} V &\leq \int_{\mathcal{X}} \|\mathbf{J}_\mathbf{f}(\mathbf{x}; \mathbf{X})\|_2^q dx_1 dx_2 \dots dx_q \\ &= \int_{\mathcal{X}} \|\mathbf{Y}\mathbf{J}_\mathbf{b}(\mathbf{x}; \mathbf{X})\|_2^q dx_1 dx_2 \dots dx_q \\ &\leq \|\mathbf{Y}\|_2^q \int_{\mathcal{X}} \|\mathbf{J}_\mathbf{b}(\mathbf{x}; \mathbf{X})\|_2^q dx_1 dx_2 \dots dx_q. \end{aligned} \tag{4.24}$$

The components of $\mathbf{J}_\mathbf{b}(\mathbf{x}; \mathbf{X})$, the Jacobian of $\mathbf{b}(\mathbf{x}; \mathbf{X})$, are given by

$$(\mathbf{J}_\mathbf{b})_{ij} = \frac{\nabla_j K(\mathbf{x} - \mathbf{x}_i)}{Np(\mathbf{x})} - \frac{K(\mathbf{x} - \mathbf{x}_i)\nabla_j p(\mathbf{x})}{Np^2(\mathbf{x})} \tag{4.25}$$

and are finite for continuously differentiable kernels and non-vanishing density. Thus, the matrix norm $\|\mathbf{J}_\mathbf{b}(\cdot)\|_2 \leq \|\mathbf{J}_\mathbf{b}(\cdot)\|_F$ is bounded. For a finite domain $\mathcal{X}$, this implies that the manifold's volume (or area for $q = 2$) is bounded, too.

4.2.2. Objective function

Now that we have defined the function model of UKR, we need to specify a suitable learning scheme. A favorable choice is the minimization of an appropriate objective

function, since the latter for example allows us to compare the results of different training runs. In correspondence to empirical risk minimization in supervised learning (Sec. 1.3.2), the UKR objective function is defined as the mean error that results from reconstructing the data vectors from their lower dimensional representations (cf. Eq. 1.14)

$$R(\mathbf{X}) = \frac{1}{N}\sum_i \|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i;\mathbf{X})\|^2 = \frac{1}{N}\|\mathbf{Y} - \mathbf{Y}\mathbf{B}(\mathbf{X})\|_F^2. \tag{4.26}$$

Here, the matrix $\mathbf{B}(\mathbf{X})$ contains the vectors of basis functions as columns, i.e.

$$\mathbf{B}(\mathbf{X}) = \big(\mathbf{b}(\mathbf{x}_1,\mathbf{X}), \mathbf{b}(\mathbf{x}_2,\mathbf{X}), \ldots, \mathbf{b}(\mathbf{x}_N,\mathbf{X})\big), \tag{4.27}$$

so the i,j-th element is given by

$$\big(\mathbf{B}(\mathbf{X})\big)_{ij} = b_i(\mathbf{x}_j;\mathbf{X}) = \frac{K(\mathbf{x}_j - \mathbf{x}_i)}{\sum_k K(\mathbf{x}_j - \mathbf{x}_k)}. \tag{4.28}$$

Please note again that for a given density kernel the reconstruction error (4.26) solely depends on the latent variables $\mathbf{X}$. Therefore, the latent variables *and* the shape of the manifold can be optimized "in one go". In contrast to many of the aforementioned algorithms, UKR does not involve alternating projection and adaption steps.

4.3. Regularization approaches

Without any form of regularization or restriction of the domain $\mathcal{X}$, the UKR reconstruction error (4.26) can be trivially minimized to $R(\mathbf{X}) = 0$. Concerning this, imagine that the low dimensional representations $\mathbf{x}_i$ are moved infinitely apart from each other, that is $\forall_{i\neq j}\|\mathbf{x}_i - \mathbf{x}_j\| \to \infty$. The kernel function $K(\cdot)$ is a density function and as such has to decrease to zero for increasing magnitudes of its argument. Therefore, in the case described above we get

$$K(\mathbf{x}_i - \mathbf{x}_j) \to \delta_{ij} K(\mathbf{0}). \tag{4.29}$$

Consequently, $b_i(\mathbf{x}_j;\mathbf{X}) \to \delta_{ij}$ and $\mathbf{B}(\mathbf{X})$ becomes the $N \times N$ identity matrix $\mathbf{I}_N$. Likewise, the reconstruction of the i-th data vector is that data vector itself, i.e. $\mathbf{f}(\mathbf{x}_i;\mathbf{X}) = \mathbf{y}_i$, so the reconstruction error (4.26) is zero. This case corresponds to the "interpolation solution" of the original principal curve (Sec. 2.5) and the exploding likelihood of the generative model for principal curves (Sec. 2.5.1).

The existence of such an undesirable minimum of the UKR reconstruction error actually reinforces our point from section 4.2.1, where we demanded that the density in the domain $\mathcal{X}$ is sufficiently high. In the interpolation case, the latent space density is only $p(\mathbf{x};\mathbf{X}) = \frac{1}{N}K(\mathbf{0})$ at the locations $\mathbf{x}_i$ and even lower inbetween.

Naturally, in practice we would like to not only prevent the worst case, but to control the balance between a low reconstruction error and a smooth manifold in a reasonable manner. The remainder of this section describes possible regularization approaches.

4.3.1. Extension of latent space

A straight-forward way to prevent the aforementioned trivial interpolation solution and to control the complexity of a UKR model is to specify a certain allowed (finite) domain

$\mathcal{X}$, e.g. a sphere of radius r. With a unit bandwidth, forcing the kernel centers $\mathbf{x}_i$ closer together implies a stronger overlapping of the kernel functions, which in turn yields a smoother regression function (cf. the supervised case in Sec. 4.1). Training of the UKR model then means solving the optimization problem

$$\text{minimize} \quad R(\mathbf{X}) = \frac{1}{N}\|\mathbf{Y} - \mathbf{YB}(\mathbf{X})\|_F^2 \quad \text{subject to} \quad \forall_i \ \|\mathbf{x}_i\| \leq r. \tag{4.30}$$

A closely related, but softer and numerically easier method is to add a penalty term to the reconstruction error (4.26) and to minimize the regularized risk

$$R_e(\mathbf{X}, \lambda) = R(\mathbf{X}) + \lambda \sum_i \|\mathbf{x}_i\|^2. \tag{4.31}$$

One may choose other forms of penalty terms, for example the general L_p-norm with $p \neq 2$. With the help of this formalism, the model complexity can be directly controlled by the pre-factor λ or the parameterization of $\mathcal{X}$, e.g. the radius r of an origin-centered sphere. However, normally one has no information about how to choose these parameters. Bigger values of λ lead to stronger overlapping of the density kernels and thus to smoother manifolds, but it is not clear how to select λ to achieve a *certain* degree of smoothness.

4.3.2. Density in latent space

As we have already seen, the UKR regression function involves an estimate of the density in latent space as a by-product. Moreover, a lower bound of that density is a sufficient condition for a finite extension of the manifold, which in turn is necessary for generalization.

Stronger overlapping of the kernel functions coincides with higher densities in latent space, which gives rise to another method for complexity control. As in the last section, the density $p(\mathbf{x})$ can be used both in a constrained optimization problem

$$\text{minimize} \ \ R(\mathbf{X}) \ \ \text{subject to} \ \ \forall_i \ p(\mathbf{x}_i) \geq \eta K(\mathbf{0}) \tag{4.32}$$

and in form of a penalty function with some pre-factor λ. To penalize low densities, a suitable regularized risk can be expressed by

$$R_p(\mathbf{X}, \lambda) = \frac{1}{N}\|\mathbf{Y} - \mathbf{YB}(\mathbf{X})\|_F^2 - \frac{\lambda}{N} \sum_i \log p(\mathbf{x}_i). \tag{4.33}$$

In case of the hard constraint (4.32), the parameter η has to be larger than $\frac{1}{N}$, so that the interpolation solution is prevented. The largest choice $\eta = 1$, however, forces all kernel centers $\mathbf{x}_i$ together, in which case the UKR manifold collapses to the mean of the data vectors: if all $\mathbf{x}_i$ are equal, then $b_i(\mathbf{x}_j) = \frac{1}{N} \ \forall i, j$ and therefore $\mathbf{f}(\mathbf{x}_i) = \frac{1}{N}\sum_j \mathbf{y}_j$. Apart from these two extremes, suitable values for η (and also λ) are hard to specify.

Compared to a regularization based on the extension of latent space, the density based regularization tends to work more locally and does not induce a certain shape of the distribution of the latent variables. On the other hand, its implementation is more complex and more prone to numerical difficulties.

4.3.3. Leave-one-out cross-validation

Perhaps the strongest feature of UKR is the ability to include leave-one-out cross-validation (LOO-CV) without additional computational cost. Instead of minimizing the reconstruction error of a UKR model including the complete dataset, in LOO-CV each data vector $\mathbf{y}_i$ has to be reconstructed without using $\mathbf{y}_i$ itself. The corresponding UKR CV-error is given by

$$R_{cv}(\mathbf{X}) = \frac{1}{N}\sum_i \|\mathbf{y}_i - \mathbf{f}_{-i}(\mathbf{x}_i;\mathbf{X})\|^2 = \frac{1}{N}\|\mathbf{Y} - \mathbf{Y}\mathbf{B}_{cv}(\mathbf{X})\|_F^2, \tag{4.34}$$

where, exactly as in the supervised case,

$$\mathbf{f}_{-i}(\mathbf{x}) = \sum_{l\neq i} \mathbf{y}_l \frac{K(\mathbf{x}-\mathbf{x}_l)}{\sum_{j\neq i} K(\mathbf{x}-\mathbf{x}_j)}. \tag{4.35}$$

For the computation of the matrix of basis functions $\mathbf{B}_{cv}$, applying LOO-CV just means zero-ing the diagonal elements before normalizing the column sums to 1. In that way, LOO-CV practically induces no computational overhead.

Minimizing $R_{cv}(\mathbf{X})$ in order to find the optimal latent variables $\mathbf{X}$ is a direct analogue to bandwidth selection using a leave-one-out criterion in classical kernel regression (cf. Sec. 4.1.1), and as such provides a favorable approach to the problem of model selection. As long as the dataset does not include repeated observations[2], LOO-CV can be used as a built-in *automatic* complexity control.

In order to gain more insight into the regularizing effect of LOO-CV, we introduce the short-hand notation $K_{ij} = K(\mathbf{x}_i - \mathbf{x}_j)$ and rewrite the i-th contribution to the CV-error by

$$\begin{aligned}
\mathbf{y}_i - \mathbf{f}_{-i}(\mathbf{x}_i;\mathbf{X}) &= \mathbf{y}_i - \sum_{j\neq i} \mathbf{y}_j \frac{K_{ij}}{\sum_{k\neq i} K_{ik}} \\
&= \mathbf{y}_i - \sum_{j\neq i} \mathbf{y}_j \frac{K_{ij}}{\sum_{k\neq i} K_{ik}} - \mathbf{y}_i \frac{K_{ii}}{\sum_{k\neq i} K_{ik}} + \mathbf{y}_i \frac{K_{ii}}{\sum_{k\neq i} K_{ik}} \\
&= \mathbf{y}_i - \sum_{j} \mathbf{y}_j \frac{K_{ij}}{\sum_{k\neq i} K_{ik}} + \mathbf{y}_i \frac{K_{ii}}{\sum_{k\neq i} K_{ik}} \\
&= \mathbf{y}_i \left(1 + \frac{K_{ii}}{\sum_{k\neq i} K_{ik}}\right) - \sum_{j} \mathbf{y}_j \frac{K_{ij}}{\sum_{k\neq i} K_{ik}}.
\end{aligned} \tag{4.36}$$

By inserting "factors" which are equal to 1, we can further rewrite the expression

$$\begin{aligned}
\mathbf{y}_i - \mathbf{f}_{-i}(\mathbf{x}_i;\mathbf{X}) &= \mathbf{y}_i \left(\frac{\sum_{k\neq i} K_{ik}}{\sum_{k\neq i} K_{ik}} + \frac{K_{ii}}{\sum_{k\neq i} K_{ik}}\right) - \sum_{j} \mathbf{y}_j \frac{K_{ij}}{\sum_{k} K_{ik}} \frac{\sum_{k} K_{ik}}{\sum_{k\neq i} K_{ik}} \\
&= \frac{\sum_{k} K_{ik}}{\sum_{k\neq i} K_{ik}} \big(\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i;\mathbf{X})\big).
\end{aligned} \tag{4.37}$$

[2] If there are repeated observations, i.e. $\mathbf{y}_i = \mathbf{y}_{i'}$, corresponding latent points $\mathbf{x}_i = \mathbf{x}_{i'}$ could be moved to an arbitrary, but isolated location without any contribution to the CV-error. Such observations $\mathbf{y}_{i'}$ should therefore be taken out as a pre-processing step. Optionally, the error in reconstructing $\mathbf{y}_i$ could then be up-weighted correspondingly.

Therefore, we can express the CV-error (4.34) in terms of the normal reconstruction error (4.26) and a multiplicative penalty

$$R_{cv}(\mathbf{X}) = \frac{1}{N} \sum_i S_i^2(\mathbf{X}) \|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i; \mathbf{X})\|^2, \tag{4.38}$$

where the penalty $S_i(\mathbf{X})$ is given by

$$S_i(\mathbf{X}) = \frac{\sum_k K_{ik}}{\sum_{k \neq i} K_{ik}} = \frac{p(\mathbf{x}_i; \mathbf{X})}{p(\mathbf{x}_i; \mathbf{X}) - \frac{1}{N} K(\mathbf{0})} = 1 + \frac{\frac{1}{N} K(\mathbf{0})}{p(\mathbf{x}_i; \mathbf{X}) - \frac{1}{N} K(\mathbf{0})}. \tag{4.39}$$

This formulation again illustrates the importance of a sufficient overlap of the density kernels: the i-th LOO-CV error contribution equals the normal error contribution scaled up by the ratio between the density $p(\mathbf{x}_i)$ and a leave-one-out density $p(\mathbf{x}_i) - \frac{1}{N} K(\mathbf{0})$.

4.4. Optimizing a UKR model

As already stated, finding optimal latent variables $\mathbf{X}$ of a UKR model involves minimization of the empirical reconstruction error (4.26), or rather a cross-validated or penalized variant of it. If the density kernel is differentiable (e.g. Gaussian or Quartic), then the minimization can be achieved by gradient descent or more elaborate gradient-based optimization algorithms. Since the reconstruction error is a sum of squares, in principle a good alternative would be utilizing nonlinear-least-squares methods like the Levenberg-Marquardt algorithm. Unfortunately, for $\mathbf{Y} \in \mathbb{R}^{d \times N}$ and $\mathbf{X} \in \mathbb{R}^{q \times N}$ the corresponding Jacobian matrix would have $dN \times qN$ elements, which quickly becomes impractical. A similar argument applies to Newton-type algorithms where a $qN \times qN$ Hessian matrix (or an approximation) would be needed. The computational complexity of conjugated gradients methods is linear in the number of parameters (qN), but as a downside they require a costly line search.

In our experiments, we have observed good results when using the RPROP algorithm (Riedmiller and Braun, 1993; Igel and Hüsken, 2000), which is traditionally applied in the world of MLP neural networks. The RPROP algorithm features an individual step size for each parameter and a simple scheme for adapting these step sizes: If an element of the gradient changes its sign between successive evaluations, the corresponding step size is reduced, otherwise, it is slightly increased. In this way, the RPROP algorithm collects second order information while still having linear complexity.

Since the objective function (4.26) is highly non-convex, it is advisable to look for some means against getting stuck in poor local minima. Fortunately, one can easily incorporate nonlinear spectral embedding methods (cf. Sec. 2.8) to find good initial sets of the latent variables. As an alternative, UKR models may be trained using a homotopy-based optimization scheme. Before describing the two approaches, we will first present the gradient of the UKR reconstruction error together with some remarks on computational complexity.

4.4.1. Gradient of the reconstruction error

In this thesis, we restrict the class of density kernels utilized in UKR to the form

$$K(\mathbf{x}_i - \mathbf{x}_j) = F(\|\mathbf{x}_i - \mathbf{x}_j\|^2), \tag{4.40}$$

where $F(\cdot)$ is at least once differentiable. For what follows, we introduce the notation

$$x_{mn} = (\mathbf{X})_{mn} = (\mathbf{x}_n)_m, \tag{4.41}$$

that is, x_{mn} denotes the m-th component ($m = 1 \dots q$) of the n-th latent variable $\mathbf{x}_n$. Using this notation, we can express the derivatives of the kernel functions by

$$\frac{\partial K_{ij}}{\partial x_{mn}} = \frac{\partial K(\mathbf{x}_i - \mathbf{x}_j)}{\partial x_{mn}} = 2F'(\|\mathbf{x}_i - \mathbf{x}_j\|^2)(x_{mi} - x_{mj})(\delta_{in} - \delta_{jn}). \tag{4.42}$$

As a companion to the matrix of basis functions $\mathbf{B}(\mathbf{X})$ with elements

$$b_{ij} = b_i(\mathbf{x}_j; \mathbf{X}) = \frac{K(\mathbf{x}_i - \mathbf{x}_j)}{\sum_k K(\mathbf{x}_k - \mathbf{x}_j)} = \frac{K_{ij}}{\sum_k K_{kj}}, \tag{4.43}$$

we now define a matrix $\mathbf{P}(\mathbf{X})$ with elements

$$p_{ij} = \frac{-2F'(\|\mathbf{x}_i - \mathbf{x}_j\|^2)}{\sum_k K(\mathbf{x}_k - \mathbf{x}_j)}. \tag{4.44}$$

As an example, assume $K(\cdot)$ to be the Gaussian kernel. In this case, we have

$$F(\|\mathbf{x}_i - \mathbf{x}_j\|^2) = F(s) = \mathcal{C} \exp\left(-\frac{1}{2}s\right) \quad \text{and} \quad F'(s) = -\frac{1}{2}F(s) \tag{4.45}$$

with a normalization factor $\mathcal{C}$ and s denoting the squared distance between two latent space vectors, so here the matrix $\mathbf{P}(\mathbf{X})$ is identical to $\mathbf{B}(\mathbf{X})$. Similarly, in case of the Quartic kernel (cf. Eq. 4.15 and 4.17), we have

$$K_Q(\mathbf{x}_i - \mathbf{x}_j) = F(\|\mathbf{x}_i - \mathbf{x}_j\|^2) = F(s) = \tilde{\mathcal{C}}\,[1-s]_+^2 \quad \text{and} \quad F'(s) = -2\tilde{\mathcal{C}}\,[1-s]_+, \tag{4.46}$$

so the elements of $\mathbf{P}(\mathbf{X})$ are given by

$$p_{ij} = 4\frac{\left[1 - \|\mathbf{x}_i - \mathbf{x}_j\|^2\right]_+}{\sum_k [1 - \|\mathbf{x}_k - \mathbf{x}_j\|^2]_+^2}. \tag{4.47}$$

Utilizing this notation, the gradient of the reconstruction error can be expressed in matrix form by

$$\frac{\partial R(\mathbf{X})}{\partial \mathbf{X}} = \frac{2}{N}\mathbf{X}\left\{\mathbf{Q} + \mathbf{Q}^T - \operatorname{diag}(\mathbf{1}^T[\mathbf{Q} + \mathbf{Q}^T])\right\}, \tag{4.48}$$

where we used further $N \times N$ matrices

$$\mathbf{Q} = \mathbf{P} * \left\{\mathbf{M} - \mathbf{1}\mathbf{1}^T[\mathbf{M} * \mathbf{B}]\right\} \tag{4.49}$$

and

$$\mathbf{M} = \mathbf{Y}^T(\mathbf{Y}\mathbf{B} - \mathbf{Y}). \tag{4.50}$$

Here, $*$ denotes the (element-wise) Schur product between matrices, and $\mathbf{1}$ is the vector of N ones. For a detailed computation please see appendix B.

Regarding the computational complexity, the most expensive parts of a gradient evaluation are the two matrix-matrix multiplications involved in calculating $\mathbf{M}$, which are $O(dN^2)$ operations. However, when using a finite support kernel function (e.g. Quartic), the matrices $\mathbf{B}$ and $\mathbf{P}$ become sparse, which can yield drastic speed-ups when utilizing a sparse linear algebra package for calculating $\mathbf{YB}$. Furthermore, the matrix $\mathbf{M}$ is only used within Schur products with $\mathbf{B}$ and $\mathbf{P}$, so many of its elements are irrelevant and thus do not have to be computed.

4.4.2. Spectral initialization

As was discussed in section 2.8, nonlinear spectral methods (e.g. LLE, Isomap) can efficiently yield low dimensional embeddings of a possibly high dimensional dataset $\mathbf{Y}$. Moreover, with the "right" setting of a neighborhood parameter, these methods can cope with very complicated structures. Since a UKR model has no parameters besides the latent variables $\mathbf{X}$ (and the fixed kernel), it is very simple to utilize the embedding of a spectral method for the purpose of initialization. Denoting the output of the spectral method by $\hat{\mathbf{X}}$, we just have to adapt the *scaling* of $\hat{\mathbf{X}}$ to UKR's (regularized) objective function – this is again analogous to selecting a suitable bandwidth[3] in supervised kernel regression. If we wish to regularize the model by LOO-CV, this means setting

$$\mathbf{X}_{init} = \mathrm{diag}(\mathbf{s}_{opt})\hat{\mathbf{X}} \quad \text{with} \quad \mathbf{s}_{opt} = \arg\min_{\mathbf{s}} R_{cv}(\mathrm{diag}(\mathbf{s})\hat{\mathbf{X}}). \tag{4.51}$$

As we optimize the CV-error only with respect to the q-dimensional scale factor $\mathbf{s}$, we can utilize simple minimization techniques and for example start with a coarse grid search. The gradient with respect to the scale is given by

$$\nabla_{\mathbf{s}} R_{cv}(\mathrm{diag}(\mathbf{s})\hat{\mathbf{X}}) = -\frac{2}{N}\mathbf{s} * \left[(\hat{\mathbf{X}} * \hat{\mathbf{X}}) \left(\mathbf{Q} + \mathrm{diag}(\mathbf{1}^T\mathbf{Q}) \right) \mathbf{1} - 2\Big(\hat{\mathbf{X}} * (\hat{\mathbf{X}}\mathbf{Q}) \Big) \mathbf{1} \right]. \tag{4.52}$$

Here, we again made use of the matrix $\mathbf{Q}$ from (4.49). Please note, however, that the underlying matrices $\mathbf{B}$ and $\mathbf{P}$ have to be calculated from the scaled latent variables $\mathbf{X} = \mathrm{diag}(\mathbf{s})\hat{\mathbf{X}}$ instead of using the unscaled embedding $\hat{\mathbf{X}}$.

Spectral methods depend (sometimes critically) on the choice of a neighborhood parameter (typically a discrete value K), so it is advisable to repeat this procedure for different values of that parameter, yielding multiple candidates $\mathbf{X}^1_{init}, \mathbf{X}^1_{init}, \ldots \mathbf{X}^n_{init}$. Then, one selects the candidate corresponding to the lowest CV-error $R_{cv}(\mathbf{X}^i_{init})$. Thus, in effect, the UKR objective function is utilized to automatically select the neighborhood parameter of the spectral method.

In addition, it can be valuable to include a PCA solution in the candidate set, since for example in case of very noisy data the nonlinear spectral method might fail. If this is the case, detected by a lower CV-error of the PCA solution as compared to the nonlinear candidates, we propose to not simply fine-tune the UKR model starting from the scale-optimized PCA solution, but to apply a homotopy method as described in the following.

4.4.3. Homotopy-based optimization

As will be demonstrated experimentally later, spectral methods do not always yield an initial set of latent variables which is already close to a sufficiently deep local minimum. To reach such a minimum regardless of the starting point, one has to apply methods for *global* optimization. A popular example of such a method is simulated annealing (see e.g. Otten and van Ginneken, 1992), where a probabilistic scheme governs the acceptance of random steps (parameter changes). With a probability depending on a decreasing "temperature" parameter, also steps corresponding to an increase of the

[3] Please recall that we fixed $h = 1$ for UKR.

objective function are accepted, and so shallow local minima can be left again. In the UKR setting with qN parameters, however, simulated annealing quickly becomes impractical – the parameter space is just too large for random steps to arrive at a deep minimum in reasonable time.

Fortunately, there are deterministic alternatives, like the deterministic annealing approach to clustering of Rose et al. (1990) (cf. Sec. 2.1.4), or more generally the class of *homotopy* or *continuation* methods (Forster, 1995; Allgower and Georg, 2003). The underlying principle of these methods is a transformation of the original objective function into a series of smoother functions with fewer local minima, up to a limit of a function which has only one easily accessible minimum[4]. As an example, consider the convolution of a function with a Gaussian of varying width (Moré and Wu, 1997). In the limit of an infinitely small width, the Gaussian becomes the Dirac distribution, and so the convolution leaves the original function unchanged. Larger widths, however, will smooth out local minima of the original function, until eventually a simple function with only one minimum remains. In order to find the global (or at least a deep local) minimum of the original function, one traces the curve of minimizers of the transformed functions starting from the smoothest one. In practice, this can be achieved by sequentially minimizing the transformed functions.

The suitability of utilizing homotopy methods in unsupervised regression problems has been pointed out by Meinicke (2000), who also provided a general approach to specifying a suitable transformation of the objective function. In particular, he chose to parametrize the class of possible regression functions such that the complexity could be controlled by a single parameter. In the limit of the lowest possible complexity, the regression manifold contracts to a single point (usually the mean of the data), and the solution can be computed trivially.

For UKR models, this scheme can easily be applied by utilizing the regularization parameters of sections 4.3.1 and 4.3.2: a density constraint $\forall_i\ p(\mathbf{x}_i) \geq K(\mathbf{0})$ forces all latent variables onto the same location, as does an extension constraint of the form $\forall_i\ \|\mathbf{x}_i\| \leq r = 0$. Similarly, also a penalty term based on extension (4.31) or density (4.30) enforces the contraction of the manifold to a single point, if only the pre-factor λ is large enough.

In the original publication of UKR (Meinicke et al., 2005), the homotopy-based optimization approach was described as the sequential optimization of UKR models with decreasing thresholds η for the density constraint in (4.32). In particular, if one wishes to fit a UKR model with automatic LOO-CV regularization, one sequentially solves

$$\mathbf{X}^m = \arg\min_{\mathbf{X}} R_{cv}(\mathbf{X}) \ , \quad \forall_i\ p(\mathbf{x}_i) \geq \eta_m K(\mathbf{0}) \tag{4.53}$$

for $m = 1 \ldots M$ and a decreasing series of thresholds $\eta_1 > \eta_2 > \cdots > \eta_{M-1} > \eta_M = 0$. The result of the m-th optimization procedure is used as the starting point for the subsequent problem, and the last "dummy" threshold $\eta_M = 0$ makes sure that

[4] In the clustering context, this limit is given by a completely fuzzy assignment, that is, all data points belong to all clusters equally. Then, all cluster centers have to move to the same position, which for a minimal quantization error has to be the mean of the data. Please note that a similar scheme was also applied for learning the "fish bowl" with the PSOM$^+$ (Sec. 3.5). There, the easily accessible minimum of the most smooth function was the PCA solution.

actually only LOO-CV determines the complexity of the manifold. In this way, the model selection of UKR is still automatic even if intermediate thresholds η_i have to be specified – the latter may influence which minimum of the objective function is found, but they do not have a conceptual influence on the complexity of the model.

From the above considerations, it is clear that also extension constraints (e.g. with an increasing radius r) and penalty terms (with decreasing pre-factors) can be utilized for the homotopy-based optimization. Actually, if for some reason the final model should not be regularized by LOO-CV, but for example by a penalty term based on the extension of latent space, it is only natural to choose the homotopy method accordingly, i.e. to use the same penalty with higher pre-factors in the homotopy steps.

Please note that the UKR homotopy procedure can not start exactly from a maximally contracted manifold, where all latent variables are equal. In this case, the gradient is zero[5], and so a gradient-based approach would be immediately stuck.

4.4.4. Projection of new data

In contrast to PCA (Sec. 2.2) or auto-encoder MLPs (Sec. 2.3), UKR does not feature an explicit functional model for mapping a new data vector $\mathbf{y}$ into the lower dimensional latent space. Rather, this mapping $\mathbf{x} = \mathbf{g}(\mathbf{y})$ is implicitly defined by the projection of $\mathbf{y}$ onto the UKR manifold $\mathcal{M}$, that is, one has to solve the optimization problem

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{f}(\mathbf{x}; \mathbf{X})\|^2 \quad \text{subject to} \quad \mathbf{x} \in \mathcal{X}. \tag{4.54}$$

Here, $\mathbf{x}^*$ denotes the optimal latent space coordinate, so the projection (the element of $\mathcal{M}$ closest to $\mathbf{y}$) is given by $\mathbf{f}(\mathbf{x}^*; \mathbf{X})$, and the projection error is $E = \|\mathbf{y} - \mathbf{f}(\mathbf{x}^*; \mathbf{X})\|^2$.

For this, however, we need a suitably defined domain $\mathcal{X}$. If $\mathcal{X}$ was already specified as part of the regularization (e.g. by an extension or a density constraint), it is of course natural to keep that specification. Otherwise, in case of regularization by LOO-CV or penalty terms, one should in hindsight set a threshold parameter η for the density in latent space, yielding the domain

$$\mathcal{X} = \{\mathbf{x} \mid p(\mathbf{x}) \geq \eta K(\mathbf{0})\}, \tag{4.55}$$

where a possible threshold is given by

$$\eta = \frac{1}{K(\mathbf{0})} \min_i p(\mathbf{x}_i). \tag{4.56}$$

This makes sure that all latent variables $\mathbf{x}_i$ of the training data lie within $\mathcal{X}$, and that the regression function is only evaluated at positions $\mathbf{x}$ that are actually supported by the data. In case of a one-dimensional latent space, an alternative simple choice of a domain is the interval spanned by the smallest and largest latent variable.

Please note that in contrast to minimizing the full UKR objective function (4.26) or its LOO-CV variant (4.34), the optimization problem (4.54) can favorably be handled by a constrained nonlinear least squares algorithm. Here, the involved Jacobian matrix has only $d \times q$ elements and the gradient of the density constraint has only q elements.

[5] This is most easily understood from looking at (4.42).

4.4.5. Summary of the procedure

In the following we summarize the overall optimization scheme for learning of the UKR model. We hereby restrict ourselves to the default case of automatic regularization via LOO-CV.

1. *Initial candidates:* Provide a set of $(n+1)$ candidate solutions $\hat{\mathbf{X}}^i_{init},\ i = 1 \ldots n+1$, including n embeddings of a nonlinear spectral method for different neighborhood parameters and a PCA solution.

2. *Scale optimization:* Find optimal scale factors $\mathbf{s}^i_{opt}$ according to

$$\mathbf{s}^i_{opt} = \arg\min_{\mathbf{s}} R_{\mathrm{cv}}(\mathrm{diag}(\mathbf{s})\hat{\mathbf{X}}^i_{init}),$$

   and set $\mathbf{X}^i_{init} = \mathrm{diag}(\mathbf{s}^i_{opt})\hat{\mathbf{X}}^i_{init}$ (cf. eq. 4.51).

3. *Candidate selection:* Pick the best candidate $\mathbf{X}_{init} = \mathbf{X}^k_{init}$ according to

$$k = \arg\min_i R_{\mathrm{cv}}(\mathbf{X}^i_{init}).$$

   Please note that in this step actually no computations are necessary, if the CV-errors resulting from the scale optimization have been stored for all candidates.

4. *Homotopy method:* If $\mathbf{X}_{init}$ corresponds to the scaled PCA solution, then apply the homotopy method as described in section 4.4.3. In particular, for a sequence of M decreasing density thresholds $\eta_1 > \eta_2 > \cdots > \eta_M = 0$ solve

$$\mathbf{X}^m = \arg\min_{\mathbf{X}} R_{cv}(\mathbf{X})\ ,\quad \forall_i\ p(\mathbf{x}_i) \geq \eta_m K(\mathbf{0})\ ,\quad m = 1 \ldots M.$$

   Set the final set of latent variables to the last result $\mathbf{X}^M$.

5. *CV-error minimization:* If in step 3) a particular nonlinear manifold model provides the best candidate solution, assume that this solution is close enough to the optimal manifold. Therefore, starting with the scaled coordinates $\mathbf{X}_{init}$, directly minimize the CV-error

$$\mathbf{X}_{final} = \arg\min_{\mathbf{X}} R_{\mathrm{cv}}(\mathbf{X}).$$

6. *Density threshold:* Select final density threshold according to

$$\eta = \frac{1}{K(\mathbf{0})} \min_i p(\mathbf{x}_i; \mathbf{X}).$$

   Please note that this step is only necessary if new data should be projected onto the manifold, or if samples should be taken from the manifold (by generating random latent position $\mathbf{x} \in \mathcal{X}$ and evaluating $\mathbf{f}(\mathbf{x}; \mathbf{X})$).

4.5. Feature space variant

As we have seen in section 1.5.4, the "kernel trick" is a convenient tool to derive possibly powerful variants of any algorithm which can be expressed in terms of inner products between the data vectors. In order to derive a feature space variant of UKR, we have to rewrite the reconstruction error (4.26) as

$$R(\mathbf{X}) = \frac{1}{N}\|\mathbf{Y} - \mathbf{Y}\mathbf{B}(\mathbf{X})\|_F^2 = \frac{1}{N}\|\mathbf{Y}\bar{\mathbf{B}}\|_F^2 = \frac{1}{N}\mathrm{tr}(\bar{\mathbf{B}}^T\mathbf{Y}^T\mathbf{Y}\bar{\mathbf{B}}) = \frac{1}{N}\mathrm{tr}(\bar{\mathbf{B}}^T\mathbf{G}\bar{\mathbf{B}}), \quad (4.57)$$

where we used the abbreviation $\bar{\mathbf{B}} = \mathbf{B}(\mathbf{X}) - \mathbf{I}$, and $\mathbf{G} \equiv \mathbf{Y}^T\mathbf{Y}$ is the Gram matrix of the data with entries $g_{ij} = \mathbf{y}_i \cdot \mathbf{y}_j$.

Naturally, also the partial derivatives can solely be computed in terms of inner products, which can be seen by differentiating (4.57)

$$\frac{\partial R(\mathbf{X})}{\partial x_{ij}} = \frac{1}{N}\frac{\partial}{\partial x_{ij}}\mathrm{tr}(\bar{\mathbf{B}}^T\mathbf{G}\bar{\mathbf{B}}) = \frac{2}{N}\mathrm{tr}\left(\bar{\mathbf{B}}^T\mathbf{G}\frac{\partial \mathbf{B}}{\partial x_{ij}}\right), \quad (4.58)$$

or alternatively by looking at the definition (4.50) of the helper matrix

$$\mathbf{M} = \mathbf{Y}^T(\mathbf{Y}\mathbf{B} - \mathbf{Y}) = (\mathbf{Y}^T\mathbf{Y})(\mathbf{B} - \mathbf{I}) = \mathbf{G}\bar{\mathbf{B}}, \quad (4.59)$$

which is used within the gradient calculation.

In this form, the computational effort of evaluating the objective function and its gradient is dominated by the matrix-matrix multiplication $\mathbf{G}\bar{\mathbf{B}}$, which is an $O(N^3)$ operation, in contrast to 2 matrix-matrix multiplications with effort $O(dN^2)$. Therefore, in case of very high-dimensional data spaces $\mathbb{R}^d$ and relatively small training sets $N < 2d$, using this formulation actually reduces the computational cost.

Replacing the dot product with a Mercer kernel $k(\mathbf{y}_i, \mathbf{y}_j)$ yields an implicit mapping $\mathbf{\Phi}(\cdot)$ from data space into some feature space $\mathcal{F}$, where dot products between feature space images can be expressed in terms of the Mercer kernel. The Gram matrix $\mathbf{G}$ is hereby replaced by a kernel matrix $\mathbf{K}(\mathbf{Y})$ with elements

$$k_{ij} = k(\mathbf{y}_i, \mathbf{y}_j) = \langle \Phi(\mathbf{y}_i), \Phi(\mathbf{y}_j) \rangle. \quad (4.60)$$

With the replacement $\mathbf{G} \to \mathbf{K}(\mathbf{Y})$ alone, the resulting feature space UKR algorithm can be utilized to fit an abstract manifold to the feature space images $\mathbf{\Phi}(\mathbf{y}_i)$. All aforementioned methods for complexity control, most notably LOO-CV, can be applied to the feature space variant without any changes.

A drawback of the feature space variant remains: Because the UKR function in a kernel feature space takes the form

$$\mathbf{f}(\mathbf{x}; \mathbf{X}) = \sum_i \Phi(\mathbf{y}_i) b_i(\mathbf{x}; \mathbf{X}), \quad (4.61)$$

in general the points of the UKR manifold in feature space cannot be given explicitly. Although several methods that can map feature space vectors back to data space have been proposed (see e.g. Bakir, Weston, and Schölkopf, 2003 and Kwok and Tsang, 2002), in the general case where an exact pre-image does not exist these methods can only yield an approximation.

Fortunately, in applications where only dot products between (4.61) and other feature space elements are needed, an explicit computation of the mapping is not necessary. Here one may compute a linear combination of dot products, instead of a dot product with a linear combination:

$$\begin{aligned}\langle \Phi(\mathbf{y}), \mathbf{f}(\mathbf{x};\mathbf{X}) \rangle &= \left\langle \Phi(\mathbf{y}), \sum_i \Phi(\mathbf{y}_i) b_i(\mathbf{x};\mathbf{X}) \right\rangle \\ &= \sum_i \langle \Phi(\mathbf{y}), \Phi(\mathbf{y}_i) \rangle \, b_i(\mathbf{x};\mathbf{X}) \\ &= \sum_i k(\mathbf{y},\mathbf{y}_i) b_i(\mathbf{x};\mathbf{X}). \end{aligned} \tag{4.62}$$

Please note that with the help of this trick also distances in feature space can be calculated, e.g. the distance between a feature space image $\Phi(\mathbf{y})$ and a point $\mathbf{f}(\mathbf{x};\mathbf{X})$ on the UKR manifold:

$$\begin{aligned}\|\Phi(\mathbf{y}) - \mathbf{f}(\mathbf{x};\mathbf{X})\|^2 &= \langle \Phi(\mathbf{y}) - \mathbf{f}(\mathbf{x};\mathbf{X}), \Phi(\mathbf{y}) - \mathbf{f}(\mathbf{x};\mathbf{X}) \rangle \\ &= \left\langle \Phi(\mathbf{y}) - \sum_i \Phi(\mathbf{y}_i) b_i(\mathbf{x}), \Phi(\mathbf{y}) - \sum_j \Phi(\mathbf{y}_j) b_j(\mathbf{x}) \right\rangle \\ &= \langle \Phi(\mathbf{y}), \Phi(\mathbf{y}) \rangle - 2 \sum_i \langle \Phi(\mathbf{y}), \Phi(\mathbf{y}_i) \rangle \, b_i(\mathbf{x}) \\ &\quad + \sum_{i,j} \langle \Phi(\mathbf{y}_i), \Phi(\mathbf{y}_j) \rangle \, b_i(\mathbf{x}) b_j(\mathbf{x}). \end{aligned} \tag{4.63}$$

If we now replace the feature space dot products by kernel functions, we get

$$\begin{aligned}\|\Phi(\mathbf{y}) - \mathbf{f}(\mathbf{x};\mathbf{X})\|^2 &= k(\mathbf{y},\mathbf{y}) - 2 \sum_i k(\mathbf{y},\mathbf{y}_i) b_i(\mathbf{x}) + \sum_{i,j} k(\mathbf{y}_i,\mathbf{y}_j) b_i(\mathbf{x}) b_j(\mathbf{x}) \\ &= k(\mathbf{y},\mathbf{y}) - 2 \sum_i k(\mathbf{y},\mathbf{y}_i) b_i(\mathbf{x}) + \mathbf{b}^T(\mathbf{x}) \mathbf{K}(\mathbf{Y}) \mathbf{b}(\mathbf{x}). \end{aligned} \tag{4.64}$$

Feature space UKR can for instance be utilized to visualize the classification boundary of a kernel-based support vector machine (SVM), where additional kernel parameters have already be chosen within the training procedure of the SVM. As another example, with the help of string or tree kernels (Vishwanathan and Smola, 2003) UKR could be utilized to fit manifolds to structured data, again with the possibility to embed the data in a 2-D latent space for visualization purposes.

Moreover, feature space UKR can also be applied if special metrics beyond the usual L_2 distance are desired for measuring the distance (the loss) between the data vectors and their reconstructions on the manifold. In a supervised setting, the definition of Mercer kernels on the basis of loss functions has been proposed by Weston, Chapelle, Elisseeff, Schölkopf, and Vapnik (2003). In the following, we derive a kernel that implicitly measures distances by the L_1-norm.

4.5.1. The L_1-norm kernel

If the Euclidean norm does not provide an adequate distance metrics for the data at hand, often because of a too large influence of outliers, it can be beneficial to measure

L_1-distances instead. UKR is explicitly based on the Euclidean norm, but by means of the kernel trick one can nonetheless effectively work with distances based on the L_1-norm. Defining a kernel function

$$k(\mathbf{y}, \mathbf{y}') = \frac{1}{2}\left(\|\mathbf{y}\|_1 + \|\mathbf{y}'\|_1 - \|\mathbf{y} - \mathbf{y}'\|_1\right) \tag{4.65}$$

yields a positive definite kernel matrix $\mathbf{K}(\mathbf{Y}) = (k(\mathbf{y}_i, \mathbf{y}_j))$ and thus implies existence of a mapping $\Phi(\mathbf{y})$ into some feature space $\mathcal{F}$. With this kernel, the usual L_2-norm of a feature space element is given by the L_1-norm of the pre-image:

$$\begin{aligned} \|\Phi(\mathbf{y})\|_2^2 &= \langle \Phi(\mathbf{y}), \Phi(\mathbf{y}) \rangle = k(\mathbf{y}, \mathbf{y}) \\ &= \frac{1}{2}(\|\mathbf{y}\|_1 + \|\mathbf{y}\|_1 - \|\mathbf{0}\|_1) = \|\mathbf{y}\|_1. \end{aligned} \tag{4.66}$$

Accordingly, the L_2 distance between two feature space images can be expressed by

$$\begin{aligned} \|\Phi(\mathbf{y}) - \Phi(\mathbf{y}')\|_2^2 &= \left\langle \Phi(\mathbf{y}) - \Phi(\mathbf{y}'), \Phi(\mathbf{y}) - \Phi(\mathbf{y}') \right\rangle \\ &= k(\mathbf{y}, \mathbf{y}) - 2k(\mathbf{y}, \mathbf{y}') + k(\mathbf{y}', \mathbf{y}') \\ &= \|\mathbf{y}\|_1 - \left(\|\mathbf{y}\|_1 + \|\mathbf{y}'\|_1 - \|\mathbf{y} - \mathbf{y}'\|_1\right) + \|\mathbf{y}'\|_1 \\ &= \|\mathbf{y} - \mathbf{y}'\|_1. \end{aligned} \tag{4.67}$$

To show that our proposed kernel is indeed positive definite (pd), and thus a mapping $\Phi(\cdot)$ exists, we rely on theorems stated by Schölkopf and Smola (2002, Sec. 2.4).

1. It is known that $k_2(\mathbf{y}, \mathbf{y}') = -\|\mathbf{y} - \mathbf{y}'\|_2^\beta$ is conditionally positive definite (cpd) for all values $0 \leq \beta \leq 2$.

2. If we consider the special case of 1-D data and $\beta = 1$, then we get the cpd kernel

$$k_1(y, y') = -|y - y'|. \tag{4.68}$$

3. A sum of cpd kernels is again cpd, so we can construct a cpd kernel

$$k_1(\mathbf{y}, \mathbf{y}') = -\|\mathbf{y} - \mathbf{y}'\|_1 = \sum_{i=1}^{d} -|y_i - y_i'|. \tag{4.69}$$

4. One can always construct a pd kernel $k(\cdot, \cdot)$ from a cpd kernel $\tilde{k}(\cdot, \cdot)$ in the following way:

$$k(\mathbf{y}, \mathbf{y}') = \frac{1}{2}\left(\tilde{k}(\mathbf{y}, \mathbf{y}') - \tilde{k}(\mathbf{y}, \mathbf{y}_0) \; - \tilde{k}(\mathbf{y}_0, \mathbf{y}') + \tilde{k}(\mathbf{y}_0, \mathbf{y}_0)\right).$$

   Setting the constant $\mathbf{y}_0 = \mathbf{0}$ and $\tilde{k} = k_1$ immediately yields the pd kernel

$$k(\mathbf{y}, \mathbf{y}') = \frac{1}{2}(\|\mathbf{y}\|_1 + \|\mathbf{y}'\|_1 - \|\mathbf{y} - \mathbf{y}'\|_1). \tag{4.70}$$

In this proof, we made use of *conditionally positive definite* kernels (Schölkopf and Smola, 2002), which are used to represent dissimilarities between data samples. This class of kernels includes the commonly used positive definite Mercer kernels, and can be utilized in kernel methods that are invariant under translation, e.g. Kernel PCA (Sec. 2.2.3). The reconstruction error of feature space UKR is solely based on distances between feature space elements and is therefore translation invariant, too.

4.6. Experiments

In order to illustrate the proposed learning scheme of section 4.4 on practical examples, we applied UKR on synthetic and real-world datasets. All experiments described here were carried out on a Pentium IV with 1.8 GHz running Linux. We utilized our own UKR toolbox for MATLAB[6], which is publicly available at http://www.sklanke.de. The toolbox includes C library functions for computation intensive parts.

The UKR optimization scheme proposed in 4.4 can easily incorporate spectral embedding methods for initialization of the UKR parameters. For the exemplary results presented here, we utilized the LLE algorithm (Sec. 2.8.2).

4.6.1. "Noisy spiral" data

To demonstrate the complete UKR learning scheme on a non-trivial toy example, we fitted a UKR curve ($q = 1$) to a sample ($N = 300$) of the "noisy spiral" distribution depicted in Fig. 4.3. The data was created by adding Gaussian noise of standard deviation $\sigma = 0.05$ in both dimensions to a sample of a spiral with two windings, its radius ranging from 0.2 to 1.2. As the latent density function of the UKR model, we chose the Gaussian kernel, and as the regularization approach, we chose LOO-CV.

For initialization (cf. step 1 in section 4.4.5) we applied the LLE algorithm based on K nearest neighbors. The smallest neighborhood parameter that leads to a connected graph is $K = 5$. Starting from this setting, we varied K up to 14 neighbors, giving rise to 9 different candidate sets of latent coordinates. As a tenth candidate, we used a PCA solution. For each candidate, we determined the optimal scaling by a coarse grid search followed by a direct search algorithm[7], according to step 2 of our optimization scheme. Figure 4.3 shows the resulting initializations and their corresponding CV-errors (4.34). For each of the ten cases we have visualized the initial UKR-manifold by evaluating the UKR function (4.18) on 500 regularly spaced points between the smallest and the largest latent coordinate.

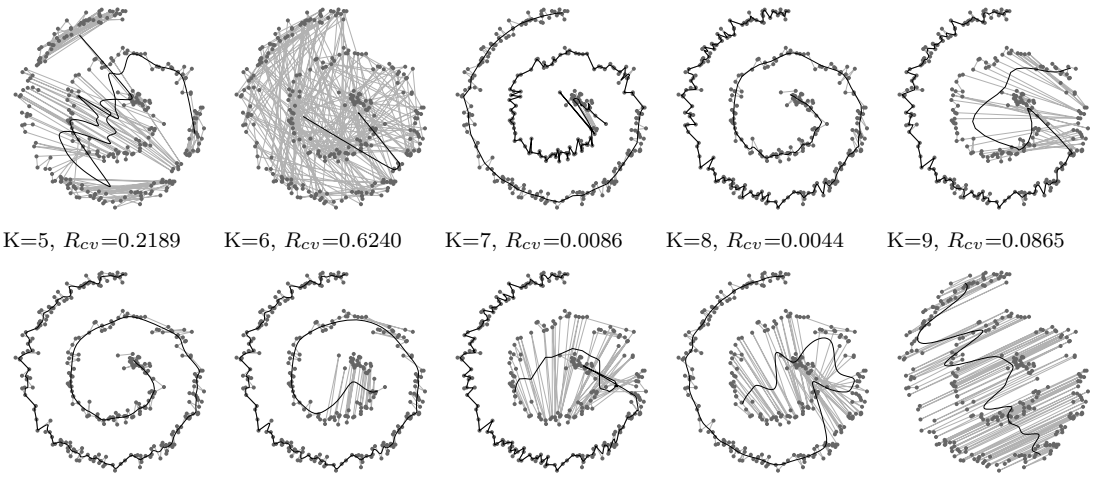
As described in step 3 of the UKR optimization scheme (Sec. 4.4.5), we selected the candidate model with the lowest CV-error, which in this case was the scaled LLE solution of neighborhood size $K = 10$. For the fine tuning (step 5) of the UKR model, we further minimized the CV-error (4.34), this time varying not only the scale, but the whole set of latent parameters $\mathbf{X}$. For this task, we used the RPROP algorithm (Riedmiller and Braun, 1993). Figure 4.4 shows how the UKR model is affected by the final gradient-based optimization. Note that the CV-error as well as the visualized manifold practically remains almost unchanged between 200 and 1000 RPROP steps.

The resulting curves in Fig. 4.4 might look undesirably unsmooth, but please keep in mind that the whole optimization and model selection procedure ran completely automatic. We did not provide the algorithm with any information about the level of noise that is present in the data, and we did not adjust any parameter by hand.

Apart from judging only by the visual impression, we investigated numerically if the

[6] MATLAB is both a programming language and an integrated environment for numerical computations, please see http://www.mathworks.com.

[7] We made use of the MATLAB function fminsearch, which implements the Nelder-Mead simplex algorithm.



K=5, R_{cv}=0.2189 K=6, R_{cv}=0.6240 K=7, R_{cv}=0.0086 K=8, R_{cv}=0.0044 K=9, R_{cv}=0.0865

K=10, R_{cv}=0.0043 K=11, R_{cv}=0.0230 K=12, R_{cv}=0.0850 K=13, R_{cv}=0.0910 PCA, R_{cv}=0.2606

Figure 4.3: Fitting UKR curves to a "noisy spiral" sample. Below each plot, the size of the LLE neighborhood (K) is given, as well as the CV-error after the scale optimization. The plot in the lower right corner shows the manifold retrieved by a PCA initialization. The black curve depicts the UKR manifold whereas the sample data is shown as dark gray dots. The initial solution (as given by LLE or PCA) is plotted in light gray. Here we just connected the data points in the order LLE/PCA places them in latent space, which corresponds to an infinite scale of the latent variable. Note that the visually best solutions ($K = 8, 10$) correspond to the smallest CV-errors.

method already tends to over-fitting in this example. In particular, we estimated the expected reconstruction error for the UKR model in the different stages of the optimization. To this end, we projected 3000 test data points sampled from the same "noisy spiral" distribution onto the UKR manifold in the four stages. For these projections we initialized the minimization problem

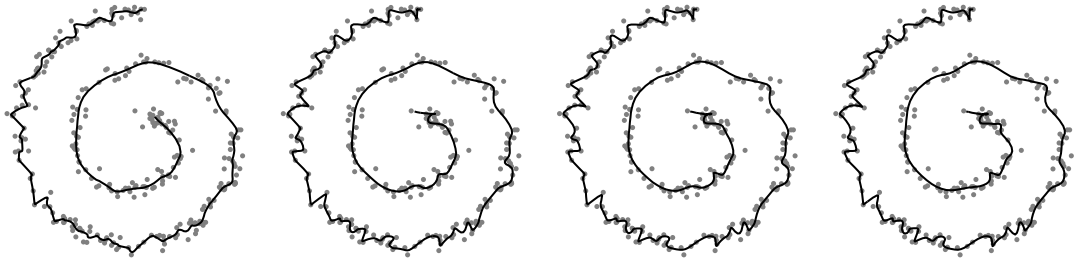
$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \|\mathbf{y} - \mathbf{f}(\mathbf{x}; \mathbf{X})\| \quad \text{s.t.} \quad p(\mathbf{x}) \geq \eta K(\mathbf{0}) \tag{4.71}$$

with the "nearest reconstruction" among the training data. That is, we chose that $\mathbf{x}_i$ among the 300 latent locations as an initial value which yields the nearest point $\mathbf{f}(\mathbf{x}_i; \mathbf{X})$ on the manifold with respect to the given test point. The density threshold η used to restrict the latent domain was selected by the heuristics described by step 6 in section 4.4.5. Table 4.1 shows the resulting errors for the different stages and illustrates that both training error (CV) *and* test error are decreasing when fine-tuning the model.

No. of steps	initialization	200	500	1000
CV-error	0.00430	0.00192	0.00185	0.00181
Proj. error	0.00289	0.00288	0.00258	0.00247

Table 4.1: Mean error for projecting 3000 test points onto the "noisy spiral" UKR manifold

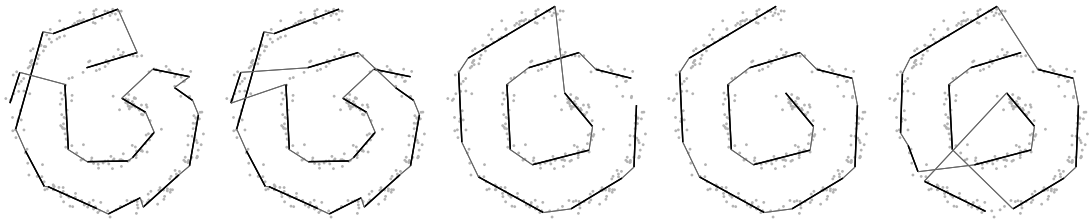
In addition to fitting a UKR model to the "noisy spiral" dataset, we also utilized the K-segments algorithm from Sec. 2.5.3 for this task. That algorithm requires to



Initialization: $R_{cv} = 0.00430$ | after 200 steps: $R_{cv} = 0.00192$ | after 500 steps: $R_{cv} = 0.00185$ | after 1000 steps: $R_{cv} = 0.00181$

Figure 4.4: Gradient-based optimization of the UKR curve on the "noisy spiral" example. The shape of the UKR manifold and the corresponding CV-error is depicted after initialization based on the best scaled LLE solution (K=10) as well as after fine-tuning by 200, 500 and 1000 RPROP steps.

set three parameters (maximum number of segments and two penalty factors), but it provides only little guidance how to select these. We tried some parameter settings by hand, and plotted the resulting[8] curves in Fig. 4.5. As can be seen, selecting successful parameters for this problem is quite hard, which is why we would like to stress again the huge advantage of the automatic model selection in UKR. In this "noisy spiral" experiment, certainly most of the credit for unwinding the underlying structure has to be assigned to the LLE algorithm. However, looking back at Fig. 4.3, it is clear that an ad-hoc choice of the neighborhood parameter would have probably failed. With UKR, we have an automatic detection of the neighborhood parameter via LOO-CV. Moreover, further fine-tuning of the model can help to correct small errors (local misplacements of the latent variables) that may be present in the spectral embedding.



$K_{max} = 15$, $\lambda = 0.1$ | $K_{max} = 15$, $\lambda = 1$ | $K_{max} = 10$, $\lambda = 10$ | $K_{max} = 10$, $\lambda = 40$ | $K_{max} = 11$, $\lambda = 40$

Figure 4.5: Some results of running the K-segments algorithm (Sec. 2.5.3) on our "noisy spiral" example. Two of the parameters of the algorithm are depicted below the plots, namely the maximal number of segments and a factor for penalizing sharp angles. The smoothing parameter σ seemed to have no effect, and was kept fixed at 0.1.

4.6.2. Homotopy and penalty terms: S-shaped triangle data

In this experiment, we would like to shed light on the homotopy method for training UKR models. At the same time, we aim at illustrating the difference between the regu-

[8] Again, we made use of the MATLAB code written by J. J. Verbeek, which is available at http://carol.wins.uva.nl/∼jverbeek/ (Mar. 2007).

larization approaches based on latent space extension and density (Sec. 4.3.1 and 4.3.2). We already stated that the density-based approach does not induce a certain shape of the distribution of latent variables, so for this experiment we chose a synthetic dataset with a characteristic shape, and investigated if this shape is recovered, or deformed.

In particular, we generated 1000 samples of a triangle in 2-D, which we then transformed onto an S-shape in 3-D space. In addition, we added Gaussian noise with standard deviation $\sigma = 0.1$. The MATLAB code for generating our data is given in Table 4.2, while the resulting dataset used in this experiment is depicted in Fig. 4.6.

```
function [Y,tt]=triangleS(N,sigma)
% Generate N samples of S-shaped triangle data with Gaussian noise (sigma)
t1 = sqrt(rand(1,N)); % aim at an even distribution within the triangle
t2 = t1.*(2*rand(1,N)-1);
x=1*sin(-1.5*pi*(2*t1-1));
y=2*t2;
z=2*sin(2.3*(2*t1-1));
Y=[x ; y; z]+sigma*randn(3,N); % combine coordinates and add noise
tt = [t1;t2];
```

Table 4.2: MATLAB code for generating our S-shaped triangle dataset.

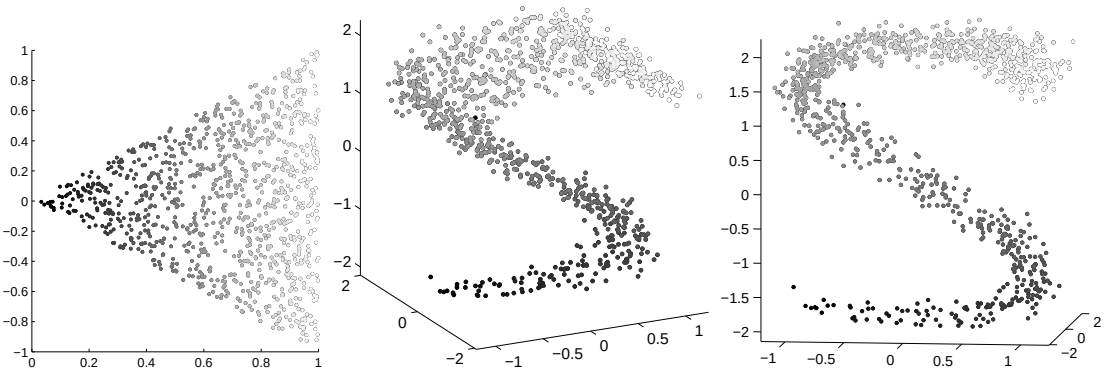


Figure 4.6: The S-shaped triangle dataset used in this experiment. Left: 1000 samples from the underlying 2-D structure. Middle and right: two alternative views on the transformed data with noise in 3-D. The gray level of the points corresponds to the generation parameter t_1 (cf. Table 4.2).

.

For simplicity, we again chose the Gaussian kernel, and calculated a PCA solution as a starting point for the homotopy-based optimization. We then carried out a series of homotopy steps with up to 500 RPROP steps each, where we minimized the sum of the reconstruction error and a penalty term based either on density (4.33) or extension (4.31). For the extension penalty, we chose the pre-factor in the k-th homotopy step as $\lambda_k = 0.7^k$, while for the density penalty we chose $\lambda_k = 0.02 \cdot 0.7^k$. The additional factor 0.02 was applied so that both penalties roughly have the same strength of effect (which we determined experimentally). Figure 4.7 shows the resulting reconstruction errors at the end of each homotopy step. We stopped the procedure as soon as the reconstruction

error dropped below 0.03 at the end of a homotopy step, where the threshold was chosen as $d\sigma^2 = 3 \cdot 0.1^2$. In this way, we explicitly made use of our knowledge about the noise level. The error threshold was reached after step 12 for the density-based approach and after step 16 for the extension-based approach.

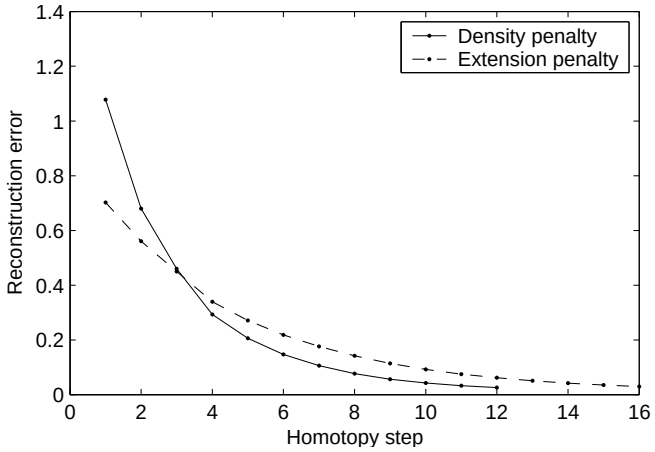


Figure 4.7: S-shaped triangle data: reconstruction errors of the UKR models at the end of each homotopy step.

Some of the intermediate UKR models based on the density penalty are depicted in Fig. 4.8, where the original data $\{\mathbf{y}_i\}$ and its UKR reconstructions $\{\mathbf{f}(\mathbf{x}_i; \mathbf{X})\}$, as well as the corresponding latent variables are plotted. Note how the manifold slowly unwinds in the first steps, and how well the triangle shape of the underlying distribution is recovered. In contrast to this, Fig. 4.9 illustrates the results of using the extension penalty term (4.31), which favors a "rounded" shape of the distribution of the latent variables and here leads to a strong bending of one of the corners of the underlying triangle. Fortunately, the UKR reconstructions $\{\mathbf{f}(\mathbf{x}_i; \mathbf{X})\}$ are not affected too strongly by this misbehavior, but the effect is visible in the 3-D plots, too. For this problem, the density based penalty clearly is the better choice.

It should be noted that our stopping rule is quite coarse, and that it cannot be applied in the general case where the noise level is unknown. Then, one should rather regularize the model automatically by utilizing LOO-CV, which can be combined with the two penalty terms for homotopy steps as well. We actually did this, too, and noted the same effect of a "bended corner" for the extension-based penalty, which was to be expected since in early steps with high pre-factors the penalty terms have much more influence than a LOO-CV regularization. On account of the focus of this experiment, however, details on the results of utilizing LOO-CV on this dataset are omitted.

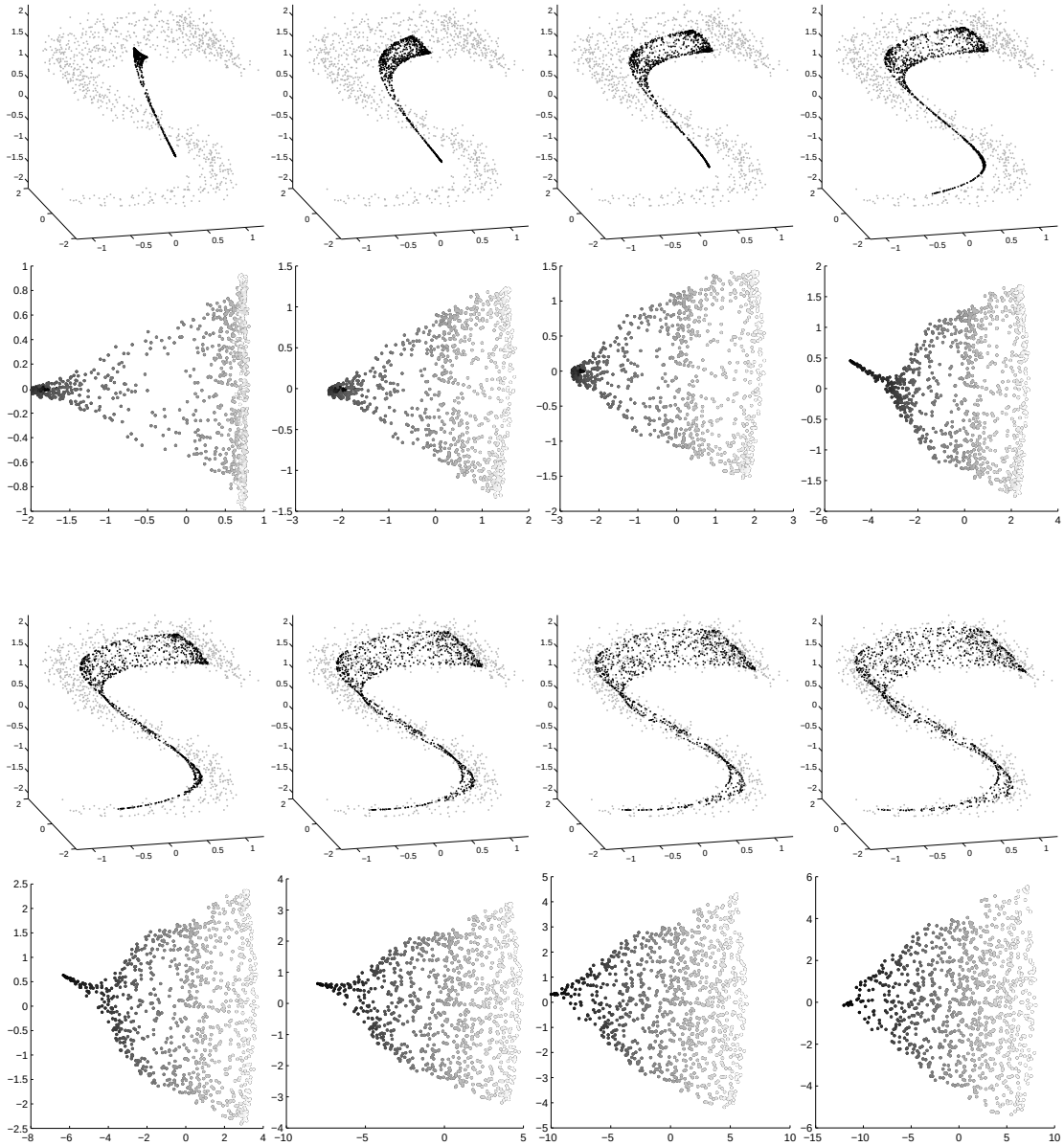


Figure 4.8: Intermediate and final solutions for the density-penalized UKR models of the S-shaped triangle dataset. In the 3-D plots, the original data $\{\mathbf{y}_i\}$ are depicted by light gray dots, while the UKR reconstructions $\{\mathbf{f}(\mathbf{x}_i; \mathbf{X})\}$ are drawn in black. Below each 3-D plot, a 2-D plot depicts the corresponding latent variables $\{\mathbf{x}_i\}$, where the color encodes one of the original generation parameters (cf. the left plot of Fig. 4.6). Please also note the varying scale of the latent variables (see the axes of the 2-D plots). From upper left to lower right, the homotopy steps $k = 1, 2, 3, 4, 6, 8, 10$ and finally 12 are shown.

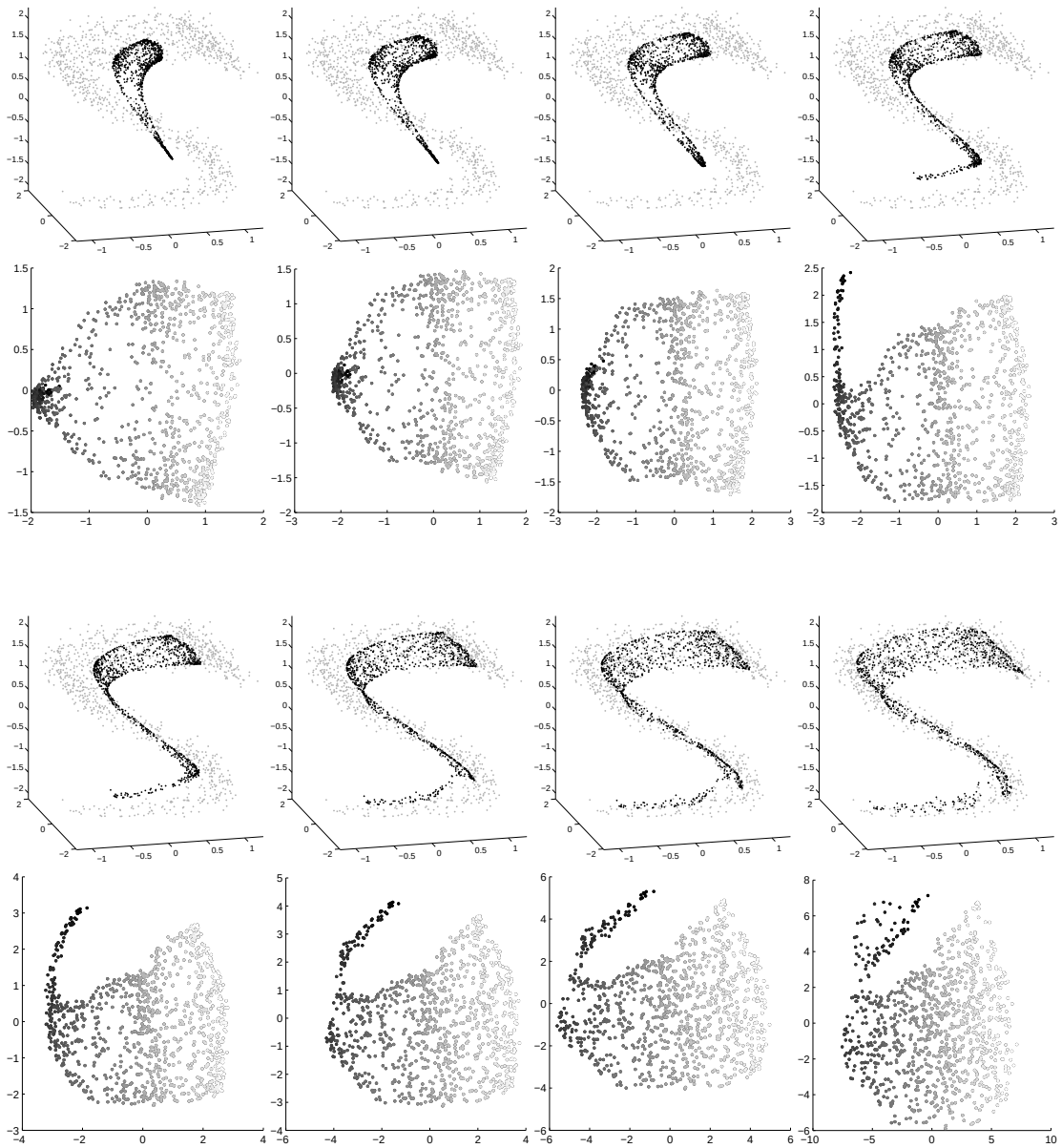


Figure 4.9: Intermediate and final solutions for the extension-penalized UKR models of the S-shaped triangle dataset. The plots are drawn in the same way as in Fig. 4.8. However, here we depicted the homotopy steps $k = 1, 2, 3, 4, 6, 9, 12$ and finally 16.

4.6.3. USPS handwritten digits

In this experiment, we applied UKR to the problem of learning suitable manifold representations for handwritten digit data. In particular we used the USPS digits dataset (see e.g. Schölkopf and Smola, 2002), which contains a total of 7291 gray-scale images of size 16x16 divided into 10 classes. Firstly, for visualization purposes, we fitted a 2-D manifold to the USPS subset representing the digit "2", which left us at 731 data vectors with 256 dimensions.

To demonstrate UKR's flexibility with respect to the choice of the latent kernel functions and to show how a finite support kernel can improve computational efficiency, we performed this part of the experiment with both the Gaussian kernel and the Quartic kernel (cf. Eq. 4.15 and 4.46). The resulting sparse structure of $\mathbf{B}(\mathbf{X})$ can lead to a significant performance gain with respect to evaluation of the error function and its gradient and is therefore particularly useful in combination with large datasets.

For fitting the model, we followed the parameter optimization scheme described in section 4.4.5. That is, we first calculated the PCA solution and several LLE solutions for a broad range of neighborhood sizes ($K = 2, 3, \ldots, 21$). Next, as in our "noisy spiral" experiment, we optimized the scale of these initialization candidates with respect to the CV-error (4.34), again utilizing a direct search method.

As before, in order to select the best candidate among our set of scaled LLE and PCA solutions, we compared the CV-errors of the different candidates. Fig. 4.10 shows these errors for both the Quartic and the Gaussian kernel. Note that both error curves are close and that the corresponding UKR models share the LLE solution with $K = 12$ nearest neighbors as the best initialization candidate (of course, each uses its own scale).

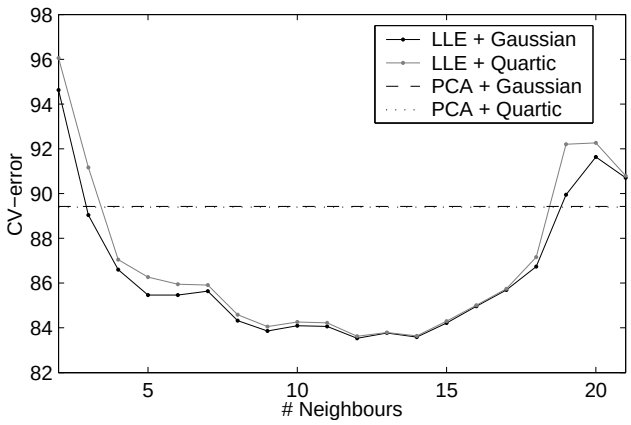
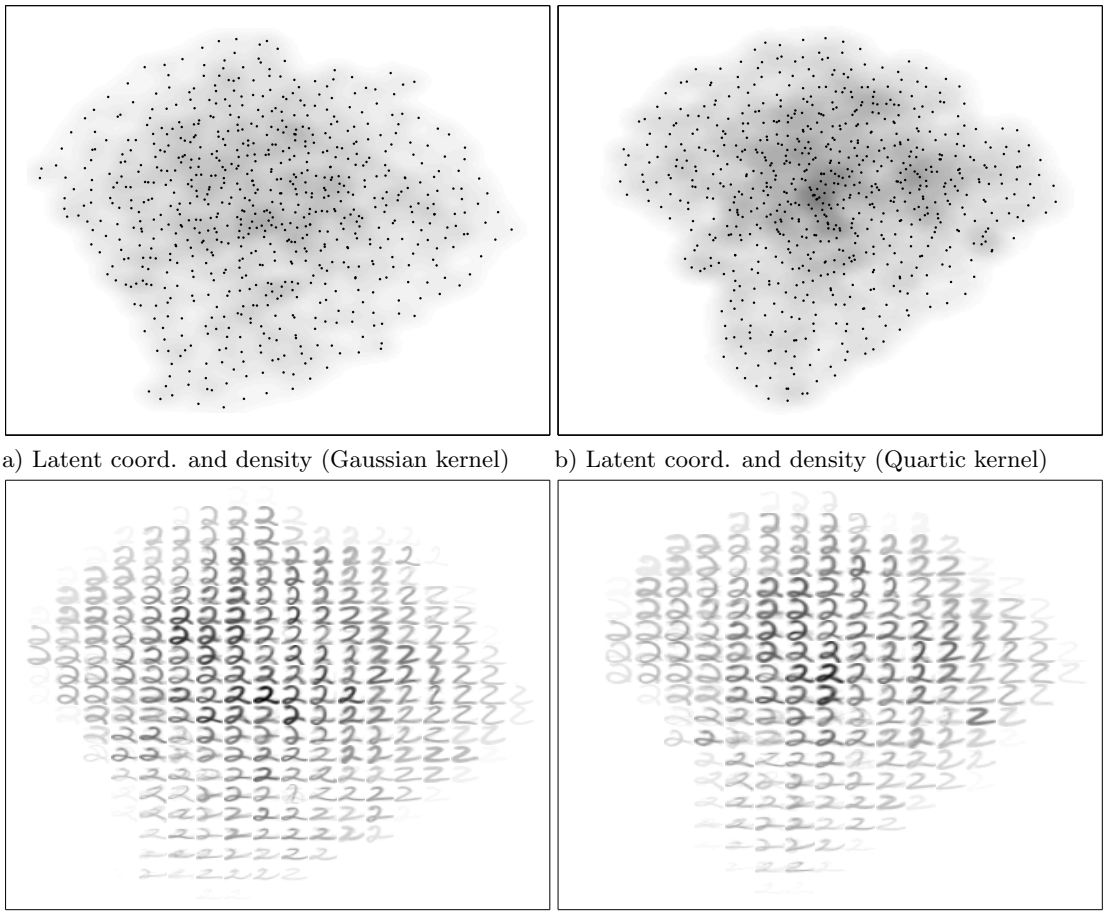


Figure 4.10: CV-errors for the different LLE and PCA solutions

As a last training step, we performed 500 RPROP steps for minimizing the CV-error with respect to the latent variables itself. At this stage, the error dropped from 83.53 to 50.90 for the Gaussian-based UKR model and from 83.62 to 51.52 for the model using the Quartic kernel in latent space.

Figure 4.11(a,b) shows the resulting latent coordinates and the corresponding density. In order to create Fig. 4.11(c,d), we sampled the latent space on a regular grid and evaluated the UKR regression function (4.18) as well as the latent density on every

node. We displayed the gray-scale images corresponding to the UKR function value and additionally scaled their intensity by the latent density at the specific grid coordinates.



a) Latent coord. and density (Gaussian kernel) b) Latent coord. and density (Quartic kernel)

c) Sampled manifold (Gaussian kernel) d) Sampled manifold (Quartic kernel)

Figure 4.11: Latent coordinates, densities and sampled manifolds of both the Gaussian and Quartic kernel based UKR model

Note that it is possible to assign meaning to the axes in the sampled manifold plots: from left to right, the digits loose their lower left bow and look more and more like a "Z". From top to bottom, the digits get thinner and flatter.

While fitting UKR models with both the Gaussian and the Quartic kernel led to similar results with respect to CV-errors and visualization, the difference in terms of CPU time is remarkable. We measured the time needed to perform 500 RPROP steps on a Linux PC with a Pentium IV processor running at 1.8 GHz and 1 GB RAM. Since the number of dimensions (256) is nearly half as big as the number of data (731), we performed this test using both normal data space UKR and feature space UKR (Sec. 4.5) with the trivial dot product kernel, that is, with a pre-calculated Gram matrix $\mathbf{G} = \mathbf{Y}^T\mathbf{Y}$. The resulting CPU times (averaged across two very close test runs) can be found in Table 4.3.

Method \ Kernel	Gaussian	Quartic
Data Space UKR	753 s	30 s
Feature Space UKR	1085 s	25 s

Table 4.3: CPU time for 500 RPROP steps on the digit "2" dataset

UKR digit classifier

In a second experiment on the USPS data, we fitted UKR models for higher dimensionalities of the latent space ($q = 5 \dots 12$) to each of the 10 digit subsets. For efficiency reasons, here we only used the Quartic kernel. Differently from the case $q = 2$, we noted that for higher dimensions LLE does not provide initial solutions that yield a lower CV-error than a coarse PCA initialization. Therefore, according to the scheme from Sec. 4.4.5, we applied the homotopy method, which we here based on density constraints. In order to account for differently sized digit subsets, we used the following 7 threshold values

$$\eta = \frac{40}{N}, \frac{32}{N}, \frac{24}{N}, \frac{16}{N}, \frac{8}{N}, \frac{4}{N} \text{ and } \frac{2}{N},$$

where N denotes the size of the respective subset, e.g. 731 for the "2". We initialized the UKR models with a PCA solution scaled down to a variance of 0.01, and for each of the seven homotopy steps we carried out 100 RPROP steps in combination with a barrier function to realize the density constraint. Finally, we minimized the pure CV-error with further 300 RPROP steps. For any given value of q, fitting 10 UKR manifolds to the 10 subsets – one manifold per digit class – roughly took an hour of computation time. Here, the increased computational effort is due to a lesser sparseness of the matrix $\mathbf{B}$ of basis functions in the earlier homotopy stages.

We then determined suitable "hindsight" thresholds η with the heuristics from (4.56), and projected the *test set* of the USPS data, containing 2007 samples in total, onto the 10 different manifolds. For each element from the test set, we determined the manifold with the smallest projection error, assigned it the corresponding class label, and compared that label with the original label from the dataset. In order to demonstrate the feature space variant of UKR with a non-trivial kernel, we repeated this experiment using the polynomial kernel of degree 3, that is

$$k(\mathbf{y}, \mathbf{y}') = (\mathbf{y} \cdot \mathbf{y}')^3, \tag{4.72}$$

as a replacement of the standard dot product. The distances between the feature space images of the test data and the UKR manifold were calculated as described in Sec. 4.5.

Table 4.4 reports the resulting error rates. Note that the best results were yielded for $q = 10$ and $q = 9$ for the standard UKR models, and those based on the polynomial kernel, respectively. Interestingly, for smaller values of q, the feature space model outperformed the standard UKR model, and vice versa. The best error rate (3.94%) is comparable to the results of other classification methods as reported by Schölkopf and Smola (2002), e.g. Kernel Fisher Discriminants (3.7%), Support Vector machines (4.0%) and Kernel PCA (4.0%). A simple nearest neighbor classifier reaches an error rate of 5.6%.

Method	$q = 5$	$q = 6$	$q = 7$	$q = 8$	$q = 9$	$q = 10$	$q = 11$	$q = 12$
Data Space UKR	4.78	4.78	4.53	4.53	4.19	3.94	3.99	4.19
Polynomial Kernel	4.43	4.58	4.19	4.24	4.14	4.29	4.24	4.33

Table 4.4: Error rates (in percent) for the UKR digit classifier for different dimensions of the latent space.

4.6.4. "Oil flow" data

For a forth and last experiment in this section, we used the "oil flow" data set, to which also the Generative Topographic Mapping (GTM, Sec. 2.6.1) and the Regularized Principal Manifolds (RPM, Sec. 2.6.2) have been applied. The corresponding results have been published by Bishop et al. (1998b) and Smola et al. (2001). The dataset consists of 12-dimensional (labeled) observations, each taking on one of three possible labels that indicate different geometric configurations inside an oil pipeline. It is subdivided into a training set, a validation set and a test set containing 1000 examples each. The intrinsic dimensionality of this dataset is known to be $q = 2$ (Bishop et al., 1998b).

Since the dataset is fairly clustered (cf. Fig. 4.12), here LLE performs poorly. As we have described in Sec. 2.8, for a spectral method to return reasonable results, one has to make sure that the neighborhood graph associated with respect to a neighborhood parameter K is connected. For the "oil flow" dataset this is the case for $K \geq 46$. Now, within neighborhoods of that size the real local properties of the data are mostly hidden, so one would hardly expect to retrieve a good mapping. As a result, all LLE solutions that we tested (we used $K = 46, 51, 56 \ldots 101$) led to substantially larger CV-errors (4.34) than a simple PCA solution. According to our optimization scheme from Sec. 4.4.5, we therefore chose the PCA solution as an initialization and used a homotopy based approach for optimization in order to avoid poor local minima.

A very similar experiment on this data can be found in the original publication of UKR (Meinicke et al., 2005), but here we applied a slightly different homotopy scheme and also used a different density kernel. In particular, we chose the Quartic kernel for computational efficiency, and we based the homotopy approach on a series of box constraints. Through this, we effectively force the latent variables to lie within a rectangular region. This does not necessarily match the underlying structure of the data, but for visualization purposes, a rectangularly shaped projection space favorably matches the typical output device (i.e. computer screen). Moreover, the latent space of the GTM, the RPM and also the PSOM$^+$ is rectangular, too, so with our choice the reader can more easily compare the 2-D projections.

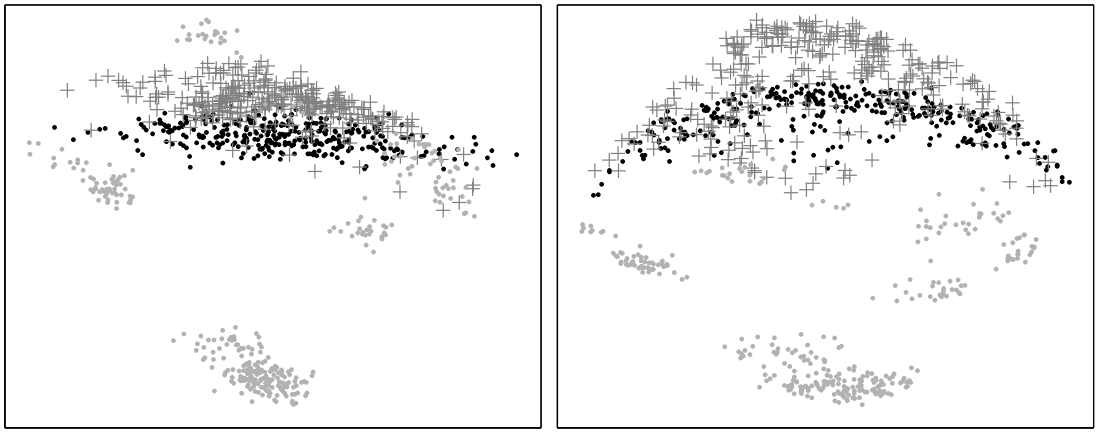
As already stated, within the homotopy steps we used a series of box constraints for the domain of the latent variables: starting from a box $\mathcal{X} = [-0.5; 0.5]^2$, we slowly increased the side length of the square by 2 in each homotopy step, that is, e.g. the second steps features a bound constraint with the box $[-1.5; 1.5]^2$. Within the steps, we minimized the CV-error (4.34) by up to 1000 steps of the RPROP algorithm, which we modified to take the box constraints into account[9]. We stopped the homotopy procedure

[9] Of all constrained optimization problems, box constraints are among the most easily implemented. One can easily clip the coordinates, determine which constraints are binding, and modify the search space accordingly.

as soon as the number of binding constraints dropped to zero, that is, in the last homotopy step we effectively minimized the unconstrained CV-error. For initialization, we scaled the PCA solution such that it fit within the initial box $[-0.5; 0.5]^2$.

To demonstrate UKR's ability to work with a different metrics, we fitted two manifolds to the training set of the "oil flow" dataset. One model was based on standard (data space) UKR, while the other model utilized feature space UKR with the L_1-kernel (4.65) proposed in section 4.5.1. For the latter case, we replaced PCA by Kernel PCA for initialization.

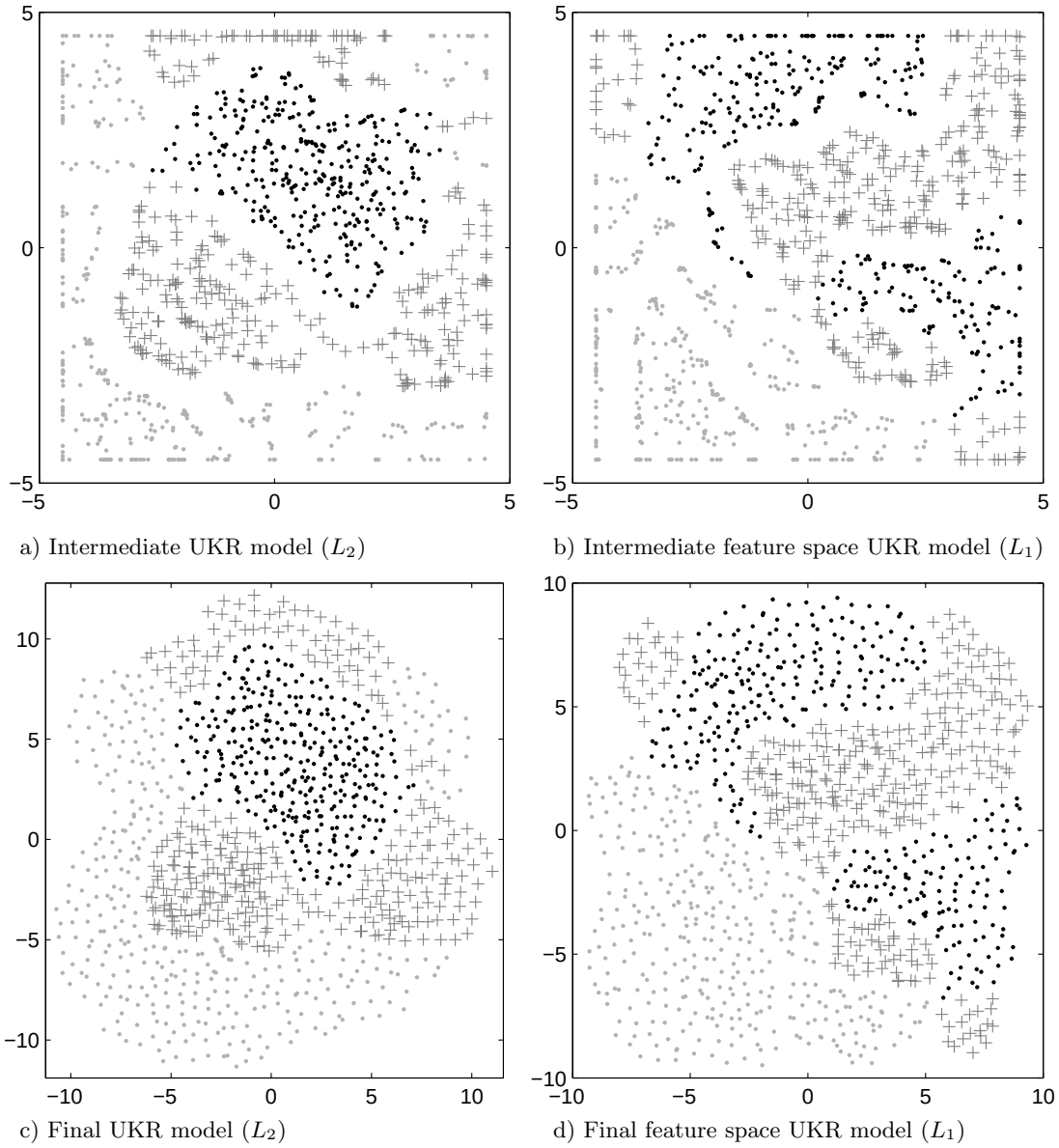
Figure 4.12 shows the results of PCA and Kernel PCA with the L_1-kernel, respectively. The data points are depicted by different symbols corresponding to their class membership. Please note that both PCA and Kernel PCA fail to separate the three classes within a 2-D projection. The results of applying UKR are shown in Fig. 4.13, which depicts an intermediate step (latent variables constrained to the box $[-4.5; 4.5]^2$) as well as the final UKR models with no binding constraints (automatically regularized by LOO-CV). Note that the separation is much improved, and that the different metrics have an effect on which classes are contiguous. Also a comparison with the 2-D projections reported by Bishop et al. (1998b) and Smola et al. (2001), regarding the separation and overlap of the different classes, seems slightly in favor of the UKR models.



a) PCA b) Kernel PCA (L_1)

Figure 4.12: Latent coordinates (2-D projections) of the "oil flow" dataset retrieved by PCA (L_2) and Kernel PCA utilizing the L_1-norm kernel.

As another comparison, we applied the PSOM$^+$ manifold learning scheme described in 3.5 to the training subset of the "oil flow" data. In particular, we utilized a Chebyshev-spaced 12x12 grid and initialized the PSOM$^+$ weights so that the initial manifold corresponds to the PCA solution. Then, we trained the PSOM$^+$ model with alternating steps of projection and weight adaption, where we again slowly relaxed the pre-factor of the roughness penalty, thus effectively using a homotopy-based optimization approach. For each homotopy step, we carried out 5 projection/adaption steps. The schedule for decreasing the pre-factor λ was chosen as $\lambda_k = 10 \cdot 0.8^k$, where k denotes the homotopy step counter.



a) Intermediate UKR model (L_2)

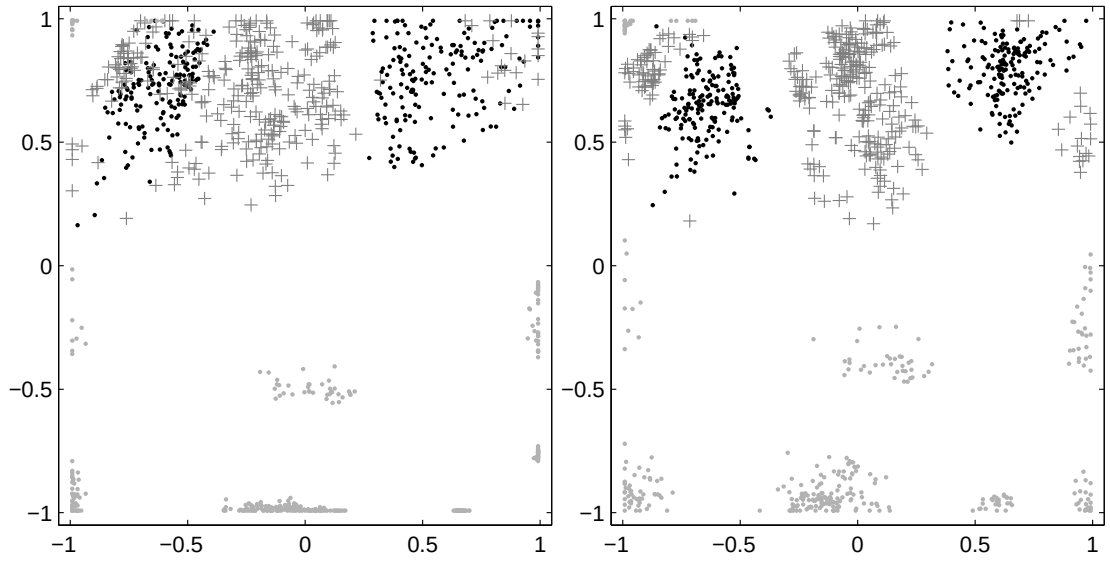
b) Intermediate feature space UKR model (L_1)

c) Final UKR model (L_2)

d) Final feature space UKR model (L_1)

Figure 4.13: Latent coordinates of "oil flow" dataset retrieved by UKR. Plots a) and c) correspond to standard data space UKR, whereas plots b) and d) belong to feature space UKR with the L_1-kernel.

For selecting the number of steps, or alternatively the final λ, we projected the *test* subset of the "oil flow" data onto the PSOM$^+$ manifold after each homotopy step, and quit the optimization procedure as soon as the test projection error increased. This was the case in the 42th step. Correspondingly, we chose $k = 41$ as the final PSOM$^+$ model, which is depicted in Fig. 4.14, as well as an intermediate solution ($k = 20$). The projections yielded by the PSOM$^+$ show quite large margins between some clusters of data points, but on the other hand some clusters overlap unfavorably (see the upper left corner of the plots in Fig. 4.14).



a) Intermediate PSOM model ($k = 20$) b) Final PSOM model ($k = 41$)

Figure 4.14: Latent coordinates (2-D projections) of the "oil flow" dataset retrieved by the PSOM$^+$ algorithm.

Regarding the computational effort, the advantage is clearly on the side of the PSOM$^+$: whereas training the UKR manifolds took 530 and 1720 seconds for the standard and feature space models, respectively, the PSOM$^+$ was trained (roughly 200 projection/adaption cycles) in only 128 seconds. However, please note that our choice of the 12x12 grid was ad-hoc, and that in general various different settings should be compared. Then, the total effort of finding a PSOM$^+$ model quickly becomes higher than training one automatically selected UKR model.

4.7. Discussion

In this chapter, we presented the UKR algorithm as an alternative approach to learning of principal manifolds. As compared with other approaches to principal curves and surfaces, UKR offers a practical way to cope with the model selection problem, since it allows us to include leave-one-out cross-validation (LOO-CV) without any additional computational cost.

Furthermore, with its non-parametric heritage from classical kernel regression, UKR obviates the need for specifying the function model beyond simply choosing a density kernel. In contrast to this, with other approaches (e.g. GTM, Sec. 2.6.1 and RPM, Sec. 2.6.2) one has to select a set of basis functions that cover the latent space in a suitable way (e.g. one basis function per node of a grid). Therefore, with UKR also higher-dimensional latent spaces can be realized with ease (cf. the USPS digit classifier, Sec. 4.6.3).

Speaking of the dimensionality q of latent space, UKR by itself provides no guidance for selecting that quantity. If no prior knowledge is available, and the application does not dictate a certain dimensionality (e.g. $q = 2$ for visualization purposes), one

should therefore first utilize other methods for estimating q. A comparison of relevant algorithms and a new approach has recently been published by Hein and Audibert (2005).

As another point in favor of UKR, no particular shape of the distribution in latent space has to be assumed, which yields great flexibility with respect to the approximation of a general data space distribution. In contrast, again, specifying a grid topology as in case of the SOM, the PSOM, or the GTM, does not necessarily match the shape of the underlying manifold well. Still, in applications where a rectangularly shaped latent space is advantageous, the UKR objective function can be combined with suitable penalty terms or extension constraints, and thus the training of the model can be influenced in that direction.

The computational effort of UKR is quite high: for standard UKR it is quadratic in the number of training samples, and it is even cubic in case of feature space UKR. Furthermore, the reconstruction error (4.26) and the CV-error (4.34) are highly non-convex, complicating optimization issues even more. Fortunately, it is very easy to initialize UKR models by means of nonlinear spectral embedding methods (cf. Sec. 4.4.2), because the latent variables $\{\mathbf{x}_i\}$ themselves are the only parameters of the UKR model. As demonstrated experimentally, UKR can even be used to automatically select the best setting of the neighborhood parameter on which any spectral method crucially depends.

Finally, by means of the kernel trick UKR can be utilized to fit principal manifolds in abstract feature spaces, possibly allowing us to work with data that have no natural representation within a usual vector space $\mathbb{R}^d$. Alternatively, feature space UKR may be useful if higher-order features of the data are of interest, as was demonstrated by the USPS digit classification experiment.

5. Extensions to Unsupervised Kernel Regression

In this chapter, we present several extensions to the original formulation of UKR. Along with the derivation of each extension, we will demonstrate its benefits experimentally. We begin with the inclusion of general loss functions into the UKR framework, replacing the squared Euclidean distance between training data and its reconstruction as an error measure. Then, we will generalize the leave-one-out method to leave-K-out cross-validation, which helps to reduce the variance of the UKR models, and leads to smoother manifolds as compared to LOO-CV. After that, we show how to reduce the computational effort of UKR by presenting a "landmark variant" similar in spirit to the landmark variants of some spectral methods (Sec. 2.8.1 and 2.8.3) and also to CCA and CDA (Sec. 2.7.3 and 2.7.4). Additionally, we carry the explicit smoothness measure from the PSOM$^+$ (Sec. 3.3.1) over to UKR. Finally, we present Unsupervised Local Polynomial Regression, which generalizes UKR by replacing the underlying Nadaraya-Watson estimator with a local polynomial regression estimator.

5.1. UKR with general loss functions

In this section, we extend UKR by introducing general cost functions[1], which allows us to make the algorithm more robust towards outliers in the data or to tune the method to specific noise models. As an example, consider a robotic hand grasping an object in different orientations, while one measures the output of a possibly large number of tactile sensors. The resulting dataset should then have an intrinsic structure and lie on or near a manifold embedded in a high dimensional space. To retrieve this manifold, it should be beneficial to ignore small errors in the dataset, namely the (known) inaccuracy of the tactile sensors. We can do exactly this by using Vapnik's ϵ-insensitive loss function with a tuned error threshold to penalize the distance between an observed data point and its reconstruction on the UKR manifold.

In the domain of supervised learning (mainly in function approximation or regression), the use of other cost functions besides the squared Euclidean error is common practice. Support Vector Machines for classification have originally been formulated with the ϵ-insensitive loss (Vapnik, 1998), and also in Support Vector Regression it is common practice to replace the L_2 error by a more general cost function (Smola, Schölkopf, and Müller, 1998b), either because it complies with a certain noise model (e.g. L_1 loss for Laplacian noise) or because of application specific needs (e.g. ϵ-insensitive loss to match tolerance levels, but also to admit sparse solutions). Also classical kernel regression has been used in conjunction with general cost functions, for example with Huber's robust

[1] We use the terms "cost functions" and "loss functions" interchangeably.

loss function (Huber, 1981; Härdle, 1990), or with the L_1 loss (Wang and Scott, 1994). Utilizing robust loss functions for nonparametric regression in conjunction with cross-validation has been investigated by Leung (2005).

In unsupervised learning, the application of cost functions other than the standard Euclidean error is rather rare. The formulation of the Regularized Principal Manifolds includes a general loss function, but the experiments of Smola et al. (2001) are based only on the squared Euclidean loss. The ϵ-insensitive loss has been successfully used with Hebbian and anti-Hebbian learning rules by Fyfe and MacDonald (2002). Recently, Alzate and Suykens (2005) investigated the combination of general loss functions with Kernel PCA.

This section is organized as follows: At first we present the loss functions that will be utilized later on, and then we show how to include these within the UKR framework. After that, we introduce a specialized optimization scheme for ϵ-insensitive UKR models. As the main part of this section, we then demonstrate the use of Huber's loss function and the ϵ-insensitive loss in a series of experiments. Finally, we briefly discuss the relation to feature space UKR, and we conclude the section with a summary on how our extension compares to the standard UKR algorithm.

Please note that the ideas and results presented in this section have already been published (Klanke and Ritter, 2006b, 2007).

5.1.1. Loss functions used in this work

The standard squared loss

The standard squared loss $L_2(r) = r^2$ or $L_2(\mathbf{r}) = \|\mathbf{r}\|^2$ has already been used throughout this thesis. It is popular mainly because of its analytic simplicity and its correspondence to a Gaussian distribution of the errors, but it has the drawback that in a sum of loss contributions $\sum_i L_2(r_i)$ a single large error or residual r can easily dominate the whole term. Therefore, utilizing the squared loss can yield suboptimal results if outliers are present in the data.

Huber's robust loss function

To make a regression model more robust against outliers, it is often beneficial to penalize residual errors r with the L_1 loss function, that is $L_1(r) = |r|$. Since this function is not differentiable at $r = 0$, it can not be utilized within gradient-based learning algorithms (e.g. training of a UKR model). However, a differentiable loss function with the same asymptotic characteristics has been introduced by Huber (1981):

$$L_H(r) = \begin{cases} \frac{1}{2\delta} r^2 & |r| < \delta \\ |r| - \frac{1}{2}\delta & |r| \geq \delta \end{cases} \tag{5.1}$$

The tradeoff point δ between the L_1 and L_2 characteristics can be adjusted for a specific application. As a multivariate generalization, one may use $L_H(\mathbf{r}) = \sum_i L_H(r_i)$, optionally with different thresholds δ per dimension. Please see the middle plot of Fig. 5.1 for a visualization of Huber's loss function for $\delta = 1$. For the experiments in this section, we always used $\delta = 0.01$, which yields a nearly pure L_1 characteristics.

The ϵ-insensitive loss function

In Support Vector Regression a popular choice is the ϵ-insensitive loss function

$$l_\epsilon(r) = \begin{cases} 0 & |r| < \epsilon \\ |r| - \epsilon & |r| \geq \epsilon \end{cases} \tag{5.2}$$

which is not differentiable at $|r| = \epsilon$. By squaring it (cf. the right plot of Fig. 5.1), we get a once-differentiable variant which asymptotically behaves like the squared Euclidean loss, but which does not penalize small residual errors. As a multi-variate generalization, one can use either

$$L_\epsilon(\mathbf{r}) = \sum_i l_\epsilon^2(r_i), \tag{5.3}$$

where the region of "cost-free" residuals has the form of a hyper-cube, or

$$L_\epsilon(\mathbf{r}) = l_\epsilon^2(\|\mathbf{r}\|), \tag{5.4}$$

where that cost-free region is a hyper-sphere with radius ϵ.

Utilizing one of these loss functions is a particularly good choice if one has information about the range of noise which is present in the data, because then one can tune the ϵ-threshold to that range.

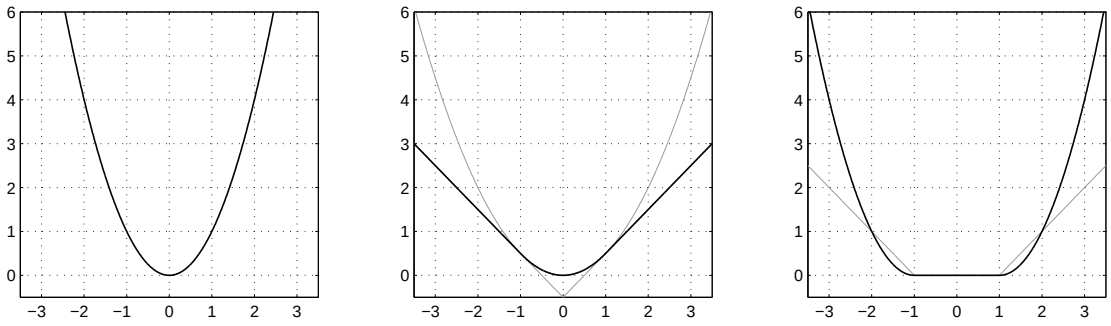


Figure 5.1: Comparison of loss functions used in this work. Left: standard L_2 loss. Middle: Huber's loss for $\delta = 1$ (black curve) and its "building blocks" (drawn gray). Right: ϵ-insensitive loss ($\epsilon = 1$) in its normal (gray) and squared form (black).

5.1.2. Including general loss functions with UKR

In order to apply the formalism of general cost functions for training UKR models, one simply replaces the squared Euclidean norm in (4.26) by the general cost function $L(\cdot)$, which yields

$$R_L(\mathbf{X}) = \frac{1}{N} \sum_i L\left(\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i; \mathbf{X})\right) = \frac{1}{N} \sum_i L(\mathbf{r}_i). \tag{5.5}$$

Since UKR training involves gradient-based minimization, the function $L(\cdot)$ has to be at least once differentiable ($L \in C^1$). Please see appendix B for the calculation of the gradient of (5.5) together with some remarks regarding the computational complexity.

Depending on the choice of the loss function, one can still automatically regularize the manifold with LOO-CV, that is, train the UKR model by minimizing the CV-error measured with a general cost function:

$$R_{L,cv}(\mathbf{X}) = \frac{1}{N} \sum_i L(\mathbf{y}_i - \mathbf{f}_{-i}(\mathbf{x}_i; \mathbf{X})). \tag{5.6}$$

However, as will be discussed in the following section, the ϵ-insensitive loss should not be applied together with LOO-CV, but rather it requires a modified learning scheme.

5.1.3. Optimization scheme for the ϵ-insensitive loss

If the squared ϵ-insensitive loss is utilized within the objective function (5.5), the UKR model is not attributed a cost as long as the reconstructions $\mathbf{f}(\mathbf{x}_i; \mathbf{X})$ lie within a certain volume element around the corresponding original data vectors $\mathbf{y}_i$. These volume elements have either the form of a hyper-cube or a hyper-sphere, depending on which multi-dimensional variant (5.3) or (5.4) is chosen.

Naturally, we would like to exploit this error tolerance in a way that the resulting UKR manifold is smoother than a corresponding manifold without tolerance. As an unfortunate consequence, it is therefore not sensible to combine the squared ϵ-insensitive loss with an automatic regularization via LOO-CV, since a wriggly and a smooth manifold that both pass through the aforementioned volume elements do not differ in terms of the UKR loss.

Thus, we are left with the regularization approaches from either section 4.3.1 or 4.3.2, of which for simplicity only the first (based on the extension of latent space) will be considered here. In principle we could add a regularization term penalizing the extension of latent space (4.31), or we can try to solve the corresponding constrained optimization problem (4.30), but either option requires the specification of a regularization parameter *in addition* to specifying a suitable ϵ.

A better variant is revealed if we once again look to the field of Support Vector Regression, where SVMs in their basic form are trained by solving the optimization problem

$$\text{minimize } \|\mathbf{w}\|^2 \quad \text{subject to} \quad |f(\mathbf{x}_i, \mathbf{w}) - y_i| < \epsilon \quad \forall i. \tag{5.7}$$

A corresponding UKR optimization problem can be stated as

$$\text{minimize } \sum_i \|\mathbf{x}_i\|^2 \quad \text{subject to} \quad L_\epsilon(\mathbf{f}(\mathbf{x}_i; \mathbf{X}) - \mathbf{y}_i) = 0 \quad \forall i, \tag{5.8}$$

or, in a more compact notation and using L_ϵ as the loss function in (5.5),

$$\text{minimize } \|\mathbf{X}\|_F^2 \quad \text{subject to} \quad R_{L_\epsilon}(\mathbf{X}) = 0. \tag{5.9}$$

In other words, we search the "smoothest" manifold (that of least possible latent extension) which passes through the ϵ-volume elements around each data vector. Thus, we can regularize the UKR model by specifying a value for ϵ, which is geometrically intuitive (as a criterion in the observed data space) and may even be pre-defined by the application (e.g. as prior knowledge about the range of noise).

We still have a problem to solve, since (5.9) might not have a solution for a given ϵ. In Support Vector Regression, this is handled by the introduction of slack variables ξ_i, which leads to the optimization problem[2]

$$\text{minimize } \|\mathbf{w}\|^2 + C\sum_i \xi_i \quad \text{subject to} \quad |f(\mathbf{x}_i, \mathbf{w}) - y_i| < \epsilon + \xi_i \quad \forall i, \tag{5.10}$$

where C balances between the rivaling goals of a maximally smooth function (smaller $\mathbf{w}$) and minimal (additional) errors ξ_i. We could introduce slack variables for UKR, too, but then we would again have to specify an additional parameter (C).

In the UKR setting, a more practical solution is to assign different error thresholds ϵ_i to each data vector (or even to each component of each data vector). For this, one first fits a possibly unsmooth manifold to the data, which is automatically regularized by LOO-CV. Then, one can measure the residual errors for each data vector, and use these residuals r_i by setting $\epsilon_i = \max(|r_i|, \epsilon)$. This is a convenient tool for reducing the influence of outliers and guarantees a valid starting point (namely the CV-regularized UKR model) of the optimization problem (5.9).

The constrained optimization itself can for example be realized by using $R_{L_\epsilon}(\mathbf{X})$ as an external penalty function for solving a series of unconstrained optimization problems

$$\text{minimize } \|\mathbf{X}\|_F^2 + \lambda R_{L_\epsilon}(\mathbf{X}) \tag{5.11}$$

with monotonously increasing values for λ (see e.g. S. S. Rao, 1978). Please note that this does not introduce a new regularization parameter, since λ goes to infinity and as such has no conceptual influence on the final model. However, a too fast schedule for increasing λ can yield suboptimal solutions because of the complex nonlinear form of the constraints.

In principle, we could also replace the extension criterion $\|\mathbf{X}\|_F^2$ by a term that favors high densities in latent space, that is we could aim at *maximizing the density* subject to the constraint $R_{L_\epsilon}(\mathbf{X}) = 0$, e.g. by solving the problem (cf. Sec. 4.3.2)

$$\text{minimize } \left(-\sum_i \log p(\mathbf{x}_i)\right) + \lambda R_{L_\epsilon}(\mathbf{X}). \tag{5.12}$$

However, while we have seen in Sec. 4.6.2 that a density-based regularization is indifferent towards the shape of the latent distribution, we do not advise to actually use a density criterion here: with the constraints having a complex nonlinear form, utilizing a non-convex objective function would only complicate numerical matters further.

5.1.4. Experiments

In this section, we demonstrate the UKR algorithm in conjunction with the loss functions we described above. All experiments were carried out on a 1.8 GHz Pentium IV running Linux, where we again used our own UKR toolbox for Matlab.

[2] In fact, to get linear constraints, one introduces two slack variables ξ_i and ξ_i^* per data point.

Noisy half circle

In our first experiment, we investigated the effect of employing Huber's loss function instead of the standard L_2 loss. As the basis for this experiment, we chose a simple "noisy half circle" toy dataset. While certainly not an impressive benchmark for curve fitting, this dataset allowed us to vary the shape and level of added noise without having to worry about bad local minima and complex initialization routines.

Firstly, we generated 100 datasets with 100 samples each, containing 1) Laplacian (biexponential) noise, the distribution of which is given by

$$p(u) = \frac{1}{2\sigma} \exp\left(-\frac{|u|}{\sigma}\right) \tag{5.13}$$

and 2) Gaussian noise, for the 4 different noise levels $\sigma = 0.25,\ 0.5,\ 0.75,\ 1$. The underlying half circle has a radius of $r = 10$. Figure 5.2 shows some examples of these datasets.

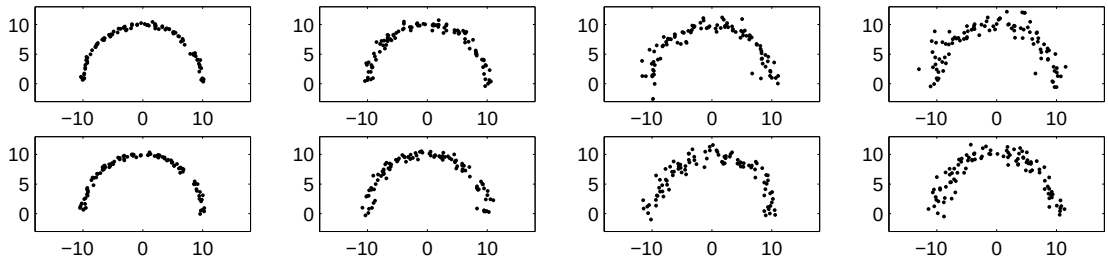


Figure 5.2: Examples of "noisy half circle" datasets, containing Laplacian noise (top row) and Gaussian noise (bottom row). From left to right: $\sigma = 0.25$, $\sigma = 0.5$, $\sigma = 0.75$, $\sigma = 1$.

We then fitted UKR models based on the Quartic kernel to each dataset. In particular, we used both the L_2 loss and Huber's loss, yielding in total 1600 models (100 datasets $\times$ 4 noise levels $\times$ 2 noise shapes $\times$ 2 loss functions). As the tradeoff point within Huber's loss function (5.1), we chose $\delta = 0.01$, resulting in an almost pure L_1 characteristics.

For initialization of the UKR models, we calculated PCA solutions which we scaled down to a variance of 1. Starting from this, we fitted the models by carrying out up to 2000 RPROP steps (Riedmiller and Braun, 1993), minimizing the LOO-CV error (5.6). These optimizations took 1.9 seconds per model on average (standard deviation 0.5 seconds).

In order to compare the UKR models resulting from the different loss functions, we measured the mean distance between the UKR reconstructions and their projections onto the underlying half circle, which in this case is simply given by

$$\bar{D} = \frac{1}{N} \sum_{i=1}^{N} |\varrho_i - 10| \quad \text{with} \quad \varrho_i = \|\mathbf{f}(\mathbf{x}_i; \mathbf{X})\|. \tag{5.14}$$

Table 5.1 summarizes the results of this experiment. Note that the UKR models trained with Huber's loss function show worse results for small noise levels, but much better results for higher noise levels as compared with the UKR models based on the L_2 loss.

The difference between the two loss functions is even more striking if one looks not only at the data reconstructions, but on the complete UKR manifold (curve) and its distance to the underlying half circle. For this, we sampled the regression function on 500 points evenly spaced between the smallest and largest latent variable. The results are displayed in Table 5.2.

Gaussian noise	org. data	standard L_2 loss	Huber's loss ($\approx L_1$)
$\sigma = 0.25$	0.201 ± 0.016	0.081 ± 0.015	0.192 ± 0.027
$\sigma = 0.50$	0.401 ± 0.027	0.171 ± 0.036	0.222 ± 0.047
$\sigma = 0.75$	0.592 ± 0.047	0.314 ± 0.067	0.256 ± 0.065
$\sigma = 1.00$	0.803 ± 0.070	0.520 ± 0.095	0.294 ± 0.077

Laplacian noise	org. data	standard L_2 loss	Huber's loss ($\approx L_1$)
$\sigma = 0.25$	0.184 ± 0.019	0.079 ± 0.018	0.187 ± 0.034
$\sigma = 0.50$	0.370 ± 0.036	0.159 ± 0.031	0.211 ± 0.046
$\sigma = 0.75$	0.547 ± 0.054	0.288 ± 0.061	0.251 ± 0.063
$\sigma = 1.00$	0.734 ± 0.059	0.459 ± 0.089	0.289 ± 0.076

Table 5.1: Mean distance between underlying half circle and the data points or their UKR reconstructions, averaged across 100 datasets (mean ± std).

Gaussian noise	standard L_2 loss	Huber's loss ($\approx L_1$)
$\sigma = 0.25$	0.088 ± 0.018	0.194 ± 0.029
$\sigma = 0.50$	0.202 ± 0.050	0.220 ± 0.045
$\sigma = 0.75$	0.386 ± 0.077	0.263 ± 0.065
$\sigma = 1.00$	0.618 ± 0.092	0.319 ± 0.081

Laplacian noise	standard L_2 loss	Huber's loss ($\approx L_1$)
$\sigma = 0.25$	0.085 ± 0.021	0.190 ± 0.035
$\sigma = 0.50$	0.197 ± 0.045	0.209 ± 0.043
$\sigma = 0.75$	0.378 ± 0.080	0.254 ± 0.058
$\sigma = 1.00$	0.587 ± 0.109	0.304 ± 0.077

Table 5.2: Mean distance between underlying half circle and the sampled UKR manifold, averaged across 100 datasets (mean ± std).

As a similar experiment, we created another 100 datasets, each containing 100 samples from the half circle again. This time, however, we randomly picked 10 samples and multiplied the corresponding 2-D vectors by a random factor uniformly distributed between 0.5 and 1.5, aimed at generating outliers. In addition, we added Gaussian noise with standard deviation $\sigma = 0.25$ to the complete dataset. See Fig. 5.3 for an example.

Again, we fitted UKR models based on the L_2 loss and Huber's loss to the datasets by starting from a PCA solution and minimizing the LOO-CV error by 2000 RPROP steps.

Figure 5.3 shows one example of the resulting UKR manifolds. Interestingly enough, even when using the L_2 loss, UKR's automatic LOO-CV regularization is sufficient to deal with single outliers, but fails if two or more outliers are close (see the left part of the half circle).

Figure 5.4 shows all datasets and all resulting manifolds on top of each other. Note the much better fit of the UKR models based on Huber's loss function (right plot). In this experiment, the mean distance between the UKR manifolds and the underlying half circle is 0.445 ± 0.132 for the standard L_2 loss, but only 0.251 ± 0.074 for Huber's loss function.

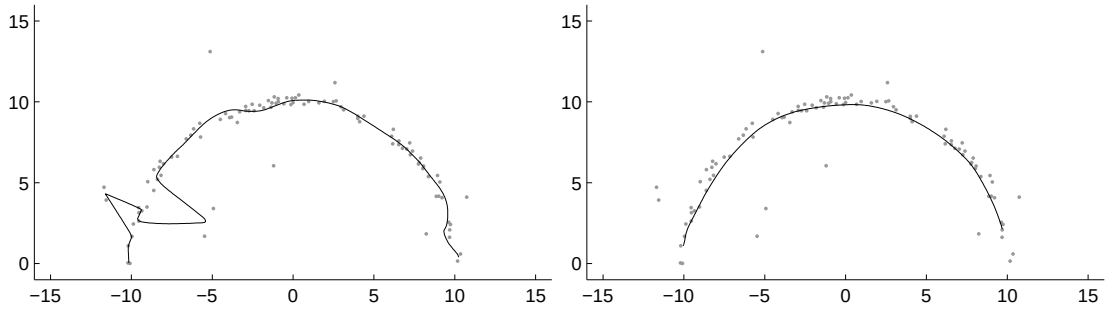


Figure 5.3: Left: UKR model based on the L_2 loss. Right: UKR model based on Huber's loss. The grey dots are the original points, the black line shows $\mathbf{f}(\mathbf{x}; \mathbf{X})$ sampled in latent space.

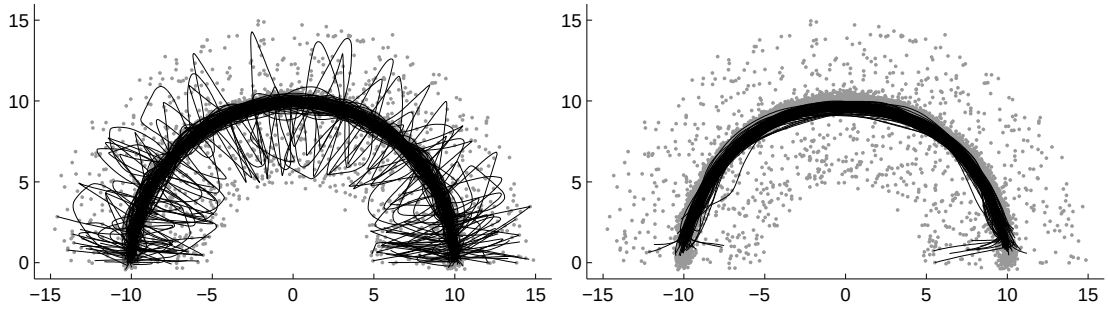


Figure 5.4: Left: UKR models based on the L_2 loss. Right: UKR models based on Huber's loss. The plots show 100 datasets and corresponding manifolds on top of each other.

Noisy spiral

As a second example, we fitted one-dimensional UKR models to a set of 100 "noisy spiral" 2-D toy datasets, which each contain 300 samples with noise distributed uniformly in the interval $[-0.1; 0.1]$. Please see the left plot of Fig. 5.5 for an example.

For initialization, we calculated Isomap (Tenenbaum et al., 2000) solutions for the 6 different neighborhood sizes $K = K_{min}, K_{min} + 1, \ldots, K_{min} + 5$, where K_{min} is the smallest neighborhood size yielding a connected graph[3], which we determined for each

[3]Please recall that for non-connected neighborhood graphs, Isomap can only yield embeddings of each component, which is not desirable here.

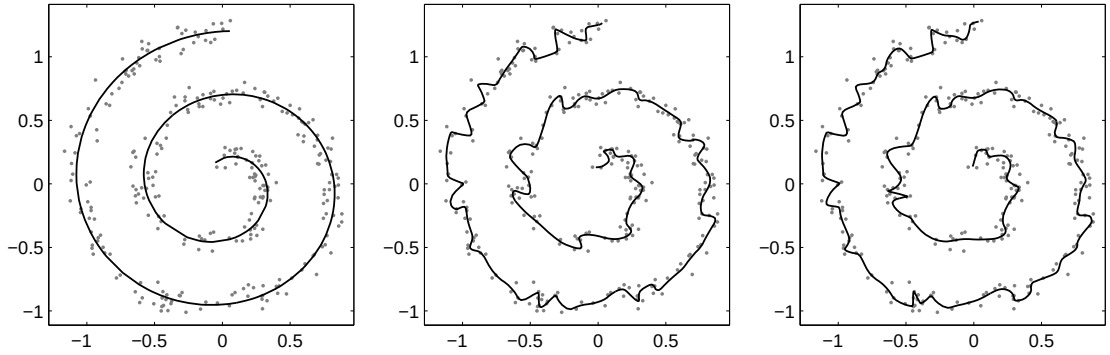


Figure 5.5: Left: underlying model of the spiral and 300 samples with uniform noise (grey dots). Middle and right: CV-regularized UKR manifold fitted to this dataset using the L_2 loss and Huber's loss. The black line shows $\mathbf{f}(\mathbf{x};\mathbf{X})$ sampled in latent space.

dataset. According to the optimization scheme of Sec. 4.4.5, we then compared the Isomap solutions with respect to their CV-error after a coarse optimization of their overall scale. Please note again that while this procedure may seem rather computationally expensive, it greatly enhances the robustness, because Isomap and other nonlinear spectral embedding methods (e.g. LLE) can depend critically on the choice of K. Figure 5.6 shows an example of Isomap and LLE solutions used as initializations of an UKR model. In this experiment we chose Isomap as a provider of initial candidates because we noted that in the presence of noise, it is more robust as compared to the LLE algorithm.

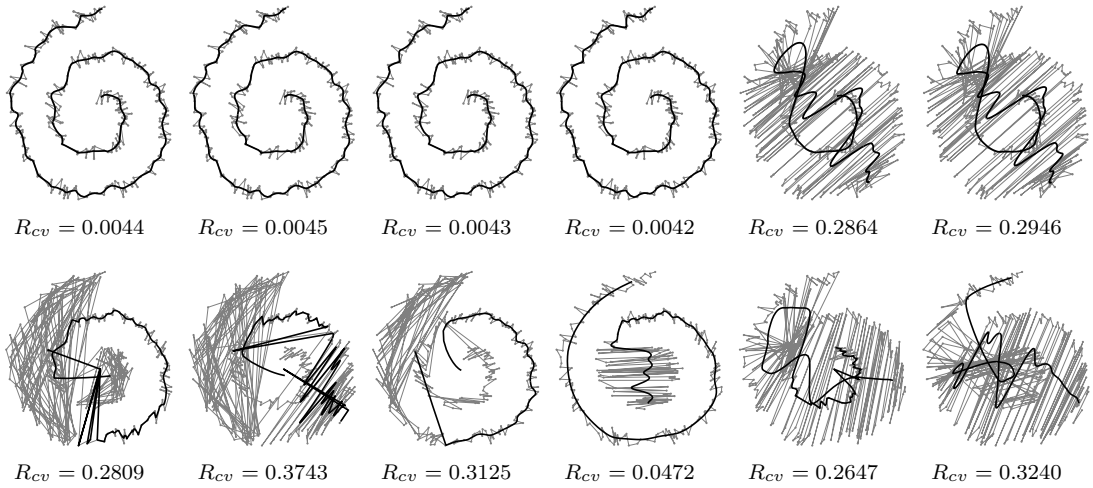


Figure 5.6: Isomap (top row) and LLE (bottom row) solutions for neighborhood sizes $K = 6 \ldots 11$ (from left to right). The data points are connected by gray lines in the order Isomap or LLE places them in the one-dimensional latent space. The black curve depicts the corresponding UKR manifold after optimization of the scale. Note that visually good solutions feature small CV-errors, as depicted below the plots.

Starting from the best candidate initialization, we first fitted CV-regularized UKR models based on the standard L_2 loss and Huber's loss by minimization of (5.6), for

Kernel function	L_2 loss	Huber's loss
Gaussian Kernel	0.0292 ± 0.0045	0.0301 ± 0.0080
Quartic Kernel	0.0285 ± 0.0050	0.0295 ± 0.0077

Table 5.3: Mean distance between UKR reconstructions and the underlying spiral, averaged across 100 datasets (mean ± std).

which we carried out 1000 RPROP steps[4]. See the middle and right plot of Fig. 5.5 for an example. We repeated this part of the experiment for both the Gaussian kernel and the Quartic kernel in latent space. For comparing the models, we again measured the mean distance between the UKR reconstructions $\mathbf{f}(\mathbf{x}_i; \mathbf{X})$ and their projection onto the underlying spiral $\mathbf{s}(t)$, that is

$$\bar{D} = \sum_{i=1}^{N} \min_t \|\mathbf{f}(\mathbf{x}_i; \mathbf{X}) - \mathbf{s}(t)\| \quad \text{where} \quad \mathbf{s}(t) = (t + 0.2) \begin{pmatrix} \sin(4\pi t) \\ \cos(4\pi t) \end{pmatrix}. \tag{5.15}$$

Table 5.3 shows the results. Note that there is nearly no difference between all four kinds of models: neither does one density kernel yield better results than the other, nor do the models of the two loss functions differ much.

However, there is a huge difference with regard to the computation time: on average, the fine-tuning ($\approx$ 1000 gradient evaluations) took 55 ± 11 seconds for the Gaussian kernel, but only 2.1 ± 0.3 seconds for the Quartic kernel.

For the rest of this experiment, we utilized the ϵ-insensitive loss function and our proposed optimization scheme from Section 5.1.1 with the aim to recover the underlying smooth spiral as close as possible. As a starting point, we used the readily trained CV-regularized UKR models based on the Quartic kernel and the L_2 loss. However, a visual inspection of these 100 models showed that five of them did not capture the spiral structure correctly. On the corresponding five datasets, Isomap failed for all 6 values of K, which is still a good result given the complexity of the problem. Since we wanted to focus on smoothing, the following experiments were carried out only for the remaining 95 "good" models.

For each CV-regularized manifold, we measured the errors made in the reconstruction of each data vector, which served as a starting point for our per-data-vector ϵ-values. We then consecutively smoothed the manifold by setting the lower bound of these values to $\epsilon = 0.01$, $\epsilon = 0.02$, $\ldots \epsilon = 0.10$. Please note that in this way, we made use of our knowledge about the noise which is present in the data: given a uniform distribution in $[-0.1; 0.1]$, allowing a larger error tolerance (a higher value of ϵ) would not make sense.

Figure 5.7 shows some of the UKR manifolds resulting from applying the constrained optimization (5.9) for one example out of the 95 datasets. Note the increasing smoothness of the curve for larger values of ϵ.

We realized the constrained optimization by using $R_{L_\epsilon}(\mathbf{X})$ as a penalty term in conjunction with the RPROP algorithm. In particular, we set the pre-factor λ in (5.11) to $\lambda = 10^2$, $\lambda = 10^4$, $\ldots \lambda = 10^{16}$ and carried out 100 unconstrained RPROP steps each.

[4]Since the scale-optimized Isomap solutions are very good initializations, we actually carry out fewer RPROP steps than for the simpler "half circle" experiment.

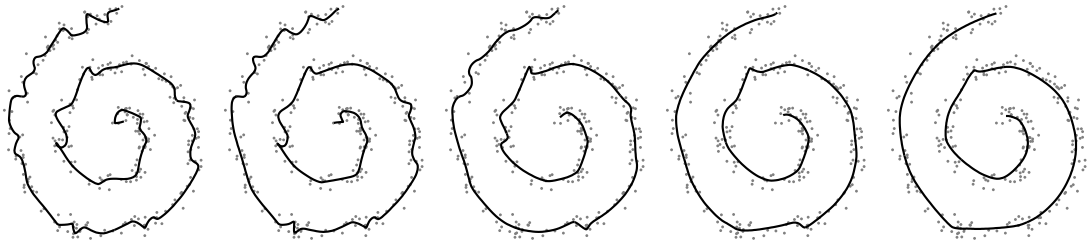


Figure 5.7: Minimal extension UKR model of a noisy spiral using the ϵ-insensitive loss function. From left to right: $\epsilon = 0.02$, $\epsilon = 0.04$, $\epsilon = 0.06$, $\epsilon = 0.08$, $\epsilon = 0.10$. The black line shows $\mathbf{f}(\mathbf{x}; \mathbf{X})$ sampled in latent space, the grey dots depict the original data vectors.

Finally, we statistically evaluated the resulting models for the 95 datasets. In particular, we sampled the manifolds on 1000 points $\mathbf{x}$ evenly spaced between the smallest and largest latent variable. We then evaluated the mean distance to the underlying spiral as in (5.15), which is depicted in the left plot of Fig. 5.8.

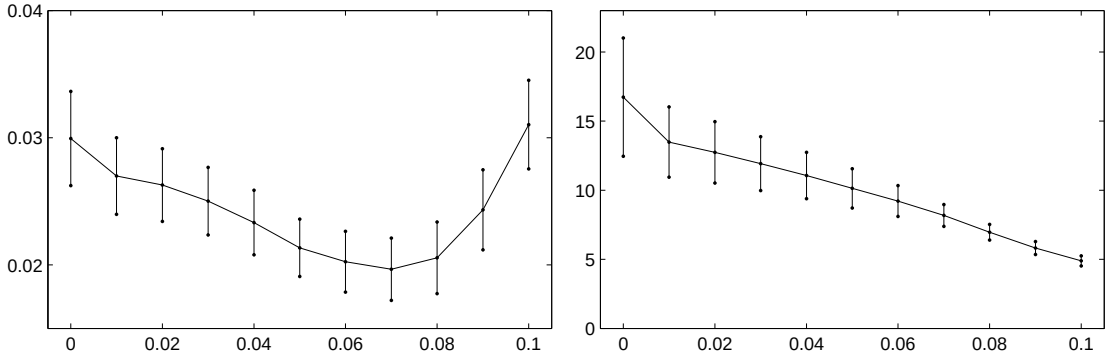


Figure 5.8: Left: mean distance between UKR manifold and underlying true spiral. Right: extension of latent space as given by $\|\mathbf{X}\|_F$. The horizontal axis corresponds to the error tolerance ϵ, whereas $\epsilon = 0$ refers to the CV-regularized models. Results are averaged across 95 datasets, with the bars depicting one standard deviation.

In addition, we also took into account whether a sampled point $\mathbf{z} = \mathbf{f}(\mathbf{x}; \mathbf{X})$ lied more towards the inner or towards the outer of the spiral. To this effect, we calculated the *signed* distance

$$\Delta = \begin{cases} +\|\mathbf{z} - \mathbf{s}(t^*)\| & \text{if } \|\mathbf{z}\| < \|\mathbf{s}(t^*)\| \\ -\|\mathbf{z} - \mathbf{s}(t^*)\| & \text{otherwise} \end{cases}, \quad t^* = \arg\min \|\mathbf{z} - \mathbf{s}(t)\|. \tag{5.16}$$

The mean $\bar{\Delta}$ of the distances for one sampled manifold yields a measure of the bias of the UKR model, whereas the standard deviation can be viewed as a measure of its smoothness (or rather roughness). Figure 5.9 reveals an increasing bias and a decreasing roughness (increasing smoothness) for bigger values of the ϵ-threshold. A good compromise between these two properties is yielded at around $\epsilon = 0.07$, where also the mean (unsigned) projection distance is significantly lower than for the CV-regularized manifolds (cf. Fig. 5.8).

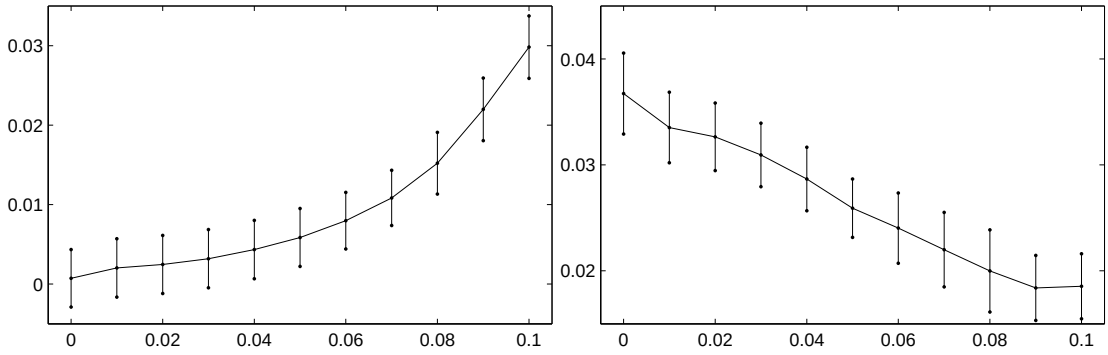


Figure 5.9: Left: mean *signed* distance $\bar{\Delta}$, with higher values indicating a bias towards the inner of the spiral. Right: standard deviation of signed distances as a roughness measure of the UKR manifold. The horizontal axis corresponds to the error tolerance ϵ, whereat $\epsilon = 0$ refers to the CV-regularized models.

Note that the existence of a bias towards the inner of the spiral is a direct consequence of the form of the UKR regression function (4.18), which calculates locally weighted convex combinations of the original data vectors. If the extension $\|\mathbf{X}\|_F$ in latent space gets smaller, there is more overlap of the kernel centers and thus the local convex combinations contain more data vectors. For the case of our spiral example (but also for e.g. the half circle) this pulls the manifold inwards. Such a bias is present (albeit negligible) already for the models regularized by LOO-CV. For ϵ-insensitive models, a larger error tolerance allows a smaller extension $\|\mathbf{X}\|_F$ and thus a larger bias. As stated above, a good compromise between smoothness and bias is yielded by setting the error tolerance ϵ to a value which is slightly smaller than the range of noise (here 0.07 compared to 0.1).

Swiss Roll

In our third experiment, we fitted two-dimensional UKR models to artifical "Swiss Roll" datasets, with the aim to investigate the effect of using Huber's loss function in a more complex learning task. We generated 100 datasets containing 1000 samples each, to which we added Gaussian noise and Laplacian noise with standard deviation $\sigma = 0.5$. Figure 5.10 shows an example of such a dataset. We then trained CV-regularized UKR models employing the standard L_2 loss and Huber's loss (with the tradeoff point set to $\delta = 0.01$), yielding in total 400 models. Because of its computational efficiency, we once more selected the Quartic kernel as the basis of the UKR models.

For initialization of our UKR models, we again used Isomap with 6 different neighborhood parameters K, starting from K_{min} (cf. the previous experiment). As before, we selected the best candidate by comparing the CV-errors (5.6) after a coarse scale optimization and then fine-tuned the UKR model by further 1000 RPROP steps. Interestingly enough, the UKR models based on the different loss functions selected different initialization candidates for 41 of the 100 datasets containing Gaussian noise and for 44 datasets containing Laplacian noise. Figure 5.11 shows an example of the resulting embeddings or latent variables. Note the more evenly spaced points in the 2-D latent

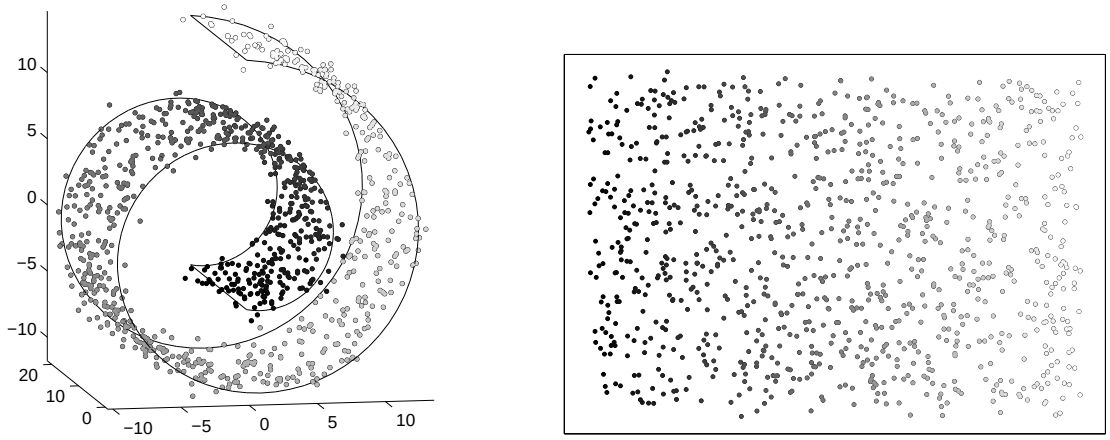


Figure 5.10: Left: one example of a "Swiss Roll" dataset containing Gaussian noise. Right: original 2-D embedding.

Noise model	org. data	L_2 loss	Huber's loss
Gaussian	0.401 ± 0.012	0.281 ± 0.020	0.236 ± 0.020
Laplacian	0.366 ± 0.011	0.266 ± 0.022	0.225 ± 0.018

Table 5.4: Mean distance between the underlying smooth "Swiss Roll" and the original data or their UKR reconstructions, respectively. Results are averaged across 100 datasets.

space as resulting from fine-tuning the UKR models (bottom row).

Regarding the average computation time, calculating an Isomap solution took 4.35 seconds per candidate, the scale optimization (a coarse grid search and a few gradient steps) took 0.47 seconds per candidate and the fine-tuning (≈1000 gradient evaluations) took 17.24 seconds. Taking 6 candidates into account, this sums up to roughly 46 seconds per model.

In order to compare the UKR models based on the two loss functions, we again measured the mean distance between the UKR reconstructions and their projection onto the underlying smooth "Swiss Roll". The results are displayed in Table 5.4. Note the significantly better results for the UKR models based on Huber's loss.

In addition, we sampled the UKR manifolds by generating 10,000 random positions inside a rectangle just large enough to contain all latent variables. To cope with embeddings that do not have the "correct" rectangular form like those in Fig. 5.11, we rejected a sample position $\mathbf{s}$ if the density $p(\mathbf{s};\mathbf{X}) < \frac{1}{2000}K(\mathbf{0})$, that is, if $\mathbf{s}$ lied in a region not populated by latent variables (or vectors) $\mathbf{x}_i$. Otherwise, we again measured the distance between $\mathbf{f}(\mathbf{s};\mathbf{X})$ and the underlying "Swiss Roll". Note that this is very similar to the definition of the UKR domain in Sec. 4.4.4, but here we used one global η for all 100 models. Table 5.5 shows the resulting mean distances, which are quite close to the results for the reconstructed data points, but with the advantage of Huber's loss function slightly diminished.

The percentage of valid sample positions is displayed in Table 5.6. Note the higher numbers for the case of Gaussian noise: this corresponds to more rectangular and thus

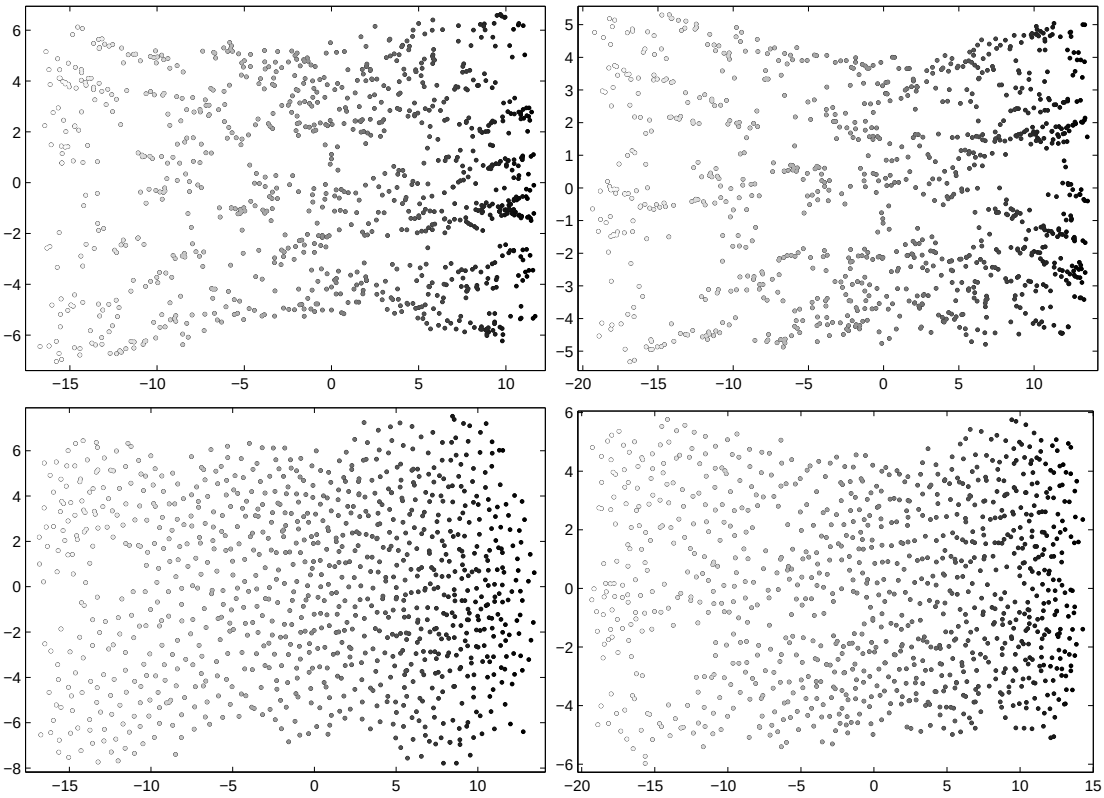


Figure 5.11: Top: selected initial set of latent variables as given by Isomap after the scale optimization. Bottom: latent variables after fine-tuning of the UKR model. The left column shows results for the UKR model based on the L_2 loss, which selected the Isomap solution with $K = 8$. The right column depicts the model based on Huber's loss, which selected a slightly different initialization candidate (belonging to $K = 7$).

more truthful embeddings, whose overall shape has to be attributed to the Isomap algorithm. Furthermore, the models utilizing Huber's loss show slightly (albeit not significantly) smaller numbers ($\approx$ less rectangular embeddings) than the L_2 models. This indicates that utilizing Huber's loss function might not be the best choice for picking initialization candidates as provided by Isomap. As a whole, this experiment demonstrates again that utilizing Huber's loss function instead of the standard L_2 loss yields favorable results.

USPS digit "2".

As a final example, we fitted a 2-D UKR manifold to the subset of the USPS handwritten digits dataset corresponding to the digit "2", which consists of 731 data vectors in 256 dimensions (16x16 pixel gray-scale images). Again, for initialization we compared several Isomap solutions with regard to their CV-error (4.34) after a coarse scale optimization. From a range of neighborhood parameters $[K_{min} = 2, 3, \ldots 16]$, $K = 2$ turned out to be optimal. We then fitted the CV-regularized UKR manifold by minimizing (4.34), carrying out 1000 RPROP steps in about 185 seconds. Then, we aimed

Noise model	L_2 loss	Huber's loss
Gaussian	0.276 ± 0.014	0.245 ± 0.019
Laplacian	0.271 ± 0.023	0.242 ± 0.024

Table 5.5: Mean distance between UKR manifold (sampled on up to 10,000 points in the 2-D latent space) and true underlying "Swiss Roll", averaged across 100 datasets.

Noise model	L_2 loss	Huber's loss
Gaussian	82.9 ± 6.7	79.2 ± 14.5
Laplacian	73.3 ± 16.7	67.7 ± 21.2

Table 5.6: Percentage of valid positions from random sampling within spanned rectangle.

at creating smooth UKR manifolds with minimal extension in latent space by solving the constrained problem (5.9).

In this dataset, most pixel values are ±1. Since for smoothing the manifold we wanted to ignore small errors in the 16x16 image as a whole and not on a pixel-by-pixel basis, we used the ϵ-insensitive loss in the sphere-shape form (5.4), that is $l_\epsilon(\|\mathbf{y} - \mathbf{f}(\mathbf{x})\|)$. Through this, the errors of all 256 data vector entries (pixels) are summed up, then the square root is taken and at last the resulting error measure is compared with the ϵ-threshold. The other way around, we can adjust the threshold to a certain number n of (completely) wrong pixels by setting $\epsilon = \sqrt{4n}$.

For fitting the ϵ-insensitive UKR manifolds, we calculated the per-data-vector ϵ-values as in the "noisy spiral" experiment, with the lower bound being set to $\epsilon = \sqrt{12}, \sqrt{24}, \ldots \sqrt{120}$, corresponding to 3,6,... 30 wrong pixels. The constrained optimization was realized just like with the "noisy spiral".

Fig. 5.12 shows the CV-regularized manifold and the manifold corresponding to 21 wrong pixels ($\epsilon = \sqrt{84}$). To create the plots, we sampled $\mathbf{f}(\mathbf{x}; \mathbf{X})$ on a grid in latent space and depicted the function value as an 16x16 image. Note the smaller extension of latent space and the blurrier images from the ϵ-optimized manifold (right plot).

Since the "correct" manifold for this dataset is unknown, quantifying the gain of using a special loss function is not as simple as in the other experiments. Here, we chose to investigate the results of projecting data from the USPS test subsets corresponding to the digits "1" (264 samples) and "2" (198 samples) onto the UKR manifold. In particular, for each of our 11 UKR models (one CV-regularized manifold and 10 different ϵ-thresholds), we measured the mean projection error $\bar{E}$ (cf. Sec. 4.4.4) of the test subset "2". After that, we counted how many of the examples from the "1" test subset could be projected with an error smaller than $\bar{E}$, which we thus viewed as false positives.

Figure 5.13 shows the results. Note the drastically reduced percentage of false positives for error tolerances of around 21 pixels. Thus, a smoother (ϵ-insensitive) manifold – provided the bias does not get too high – yields images $\mathbf{f}(\mathbf{x}; \mathbf{X})$ which are more prototypical of the dataset and hide some of the variation (or noise). This interpretation is also supported from the visual impression provided by Fig. 5.12.

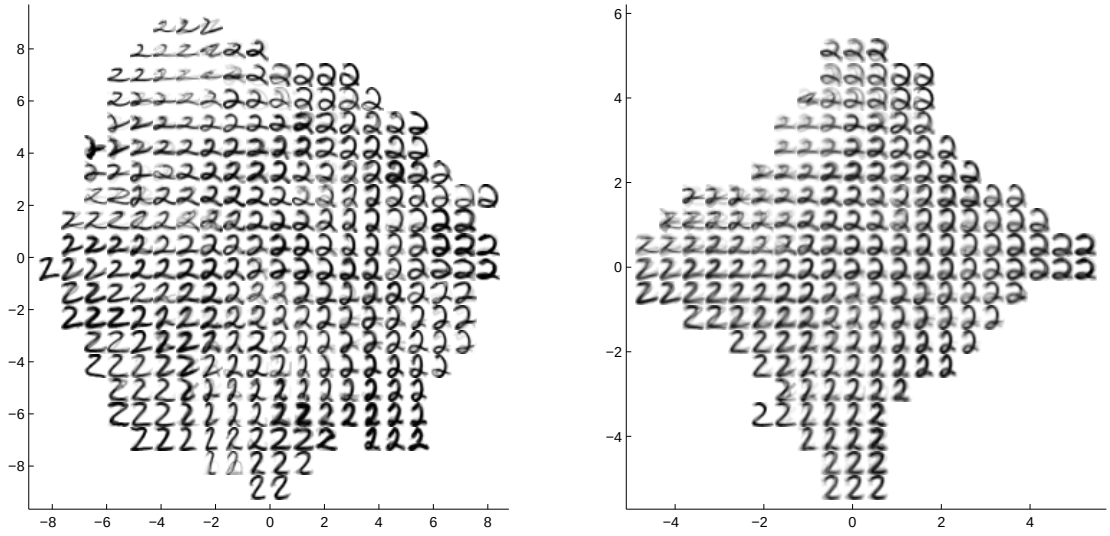


Figure 5.12: UKR model of the USPS digit "2", shown by evaluating $\mathbf{f}(\mathbf{x};\mathbf{X})$ on a 20x20 grid enclosing the latent variables. Grid positions of low density $p(\mathbf{x})$ are left blank. Left: CV regularized model. Right: ϵ-insensitive model (threshold corresponding to 21 wrong pixels).

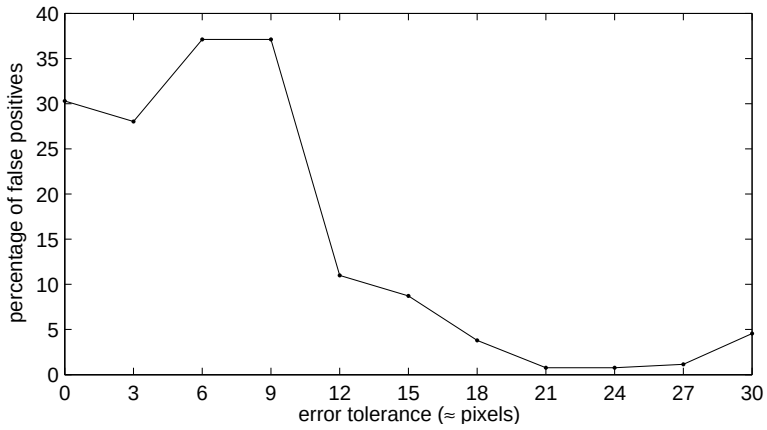


Figure 5.13: Result of projecting images of "1"s onto UKR models fitted to the "2" subset using different ϵ-thresholds. The graph shows the percentage of false positives ("1"s with a projection error smaller than $\bar{E}$) as a function of the error threshold used to train the models (here given in units of pixels). The CV-regularized manifold is depicted at x-coordinate 0.

5.1.5. Relation to feature space UKR

As another way of incorporating general and even non-differentiable loss functions, one might consider using *feature space* UKR (Sec. 4.5), where the basic idea is to construct a Mercer kernel that implicitly maps the data to some feature space so that the standard Euclidean norm in that space coincides with the demanded loss function in data space. As an example we presented the L_1-norm kernel in section 4.5.1.

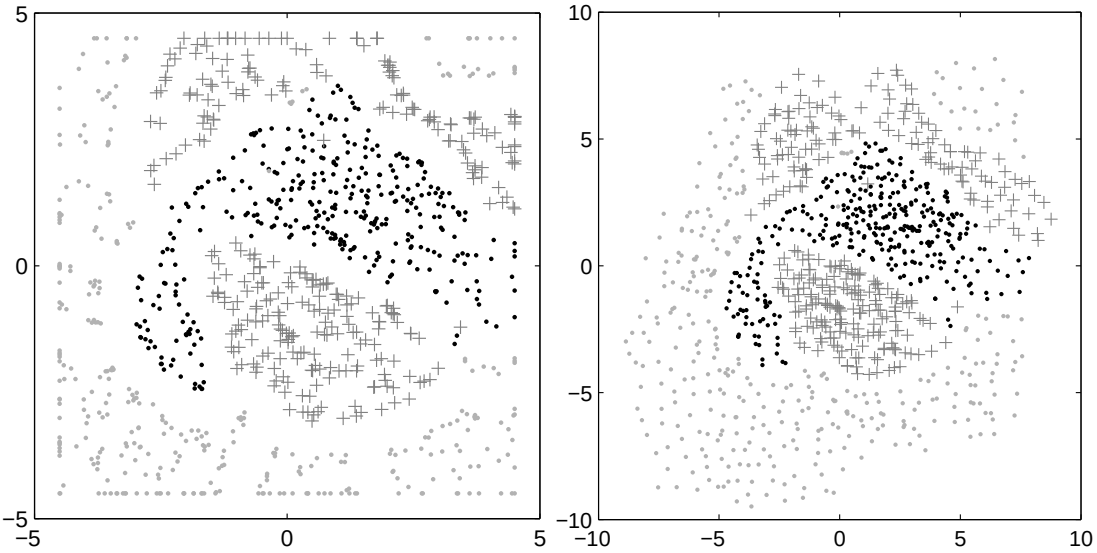
However, we have also already discussed the main problem with feature space UKR: the corresponding regression function $\mathbf{f}(\mathbf{x};\mathbf{X}) = \sum_i \Phi(\mathbf{y}_i) b_i(\mathbf{x};\mathbf{X})$ now linearly com-

bines feature space images $\Phi(\mathbf{y}_i)$ and cannot be calculated explicitly for general Mercer kernels. Furthermore, there is no guaranteed existence of a data space pre-image

$$\mathbf{y} \stackrel{?}{=} \Phi^{-1}\left(\mathbf{f}(\mathbf{x};\mathbf{X})\right) = \Phi^{-1}\left(\sum_i \Phi(\mathbf{y}_i) b_i(\mathbf{x};\mathbf{X})\right) \tag{5.17}$$

of such linear combinations, so the demanded loss function (e.g. $\|\mathbf{y} - \mathbf{y}'\|_1$ in case of the L_1-norm kernel) may have no well-defined operand. These difficulties limit the application scope of feature space UKR, and make general loss functions in the way of (5.5) the favorable choice in practice.

Moreover, with the above considerations on the pre-image problem, we must conclude that using the L_1 loss function (putting aside the problem of non-continuous derivates) is *not* equivalent to utilizing the L_1-norm kernel. Indeed, a 2-D embedding of the "oil flow" dataset (cf. Sec. 4.6.4) resulting from a UKR model with Huber's loss function[5] differs quite much from the L_1 feature space UKR model, despite having used the same initialization and optimization scheme. Please see Fig. 5.14(a,b) for the UKR model using Huber's loss, and Fig. 4.13(b,d) on page 98 for the UKR model based on the L_1-kernel.



a) Intermediate UKR model (Huber loss) b) Final UKR model (Huber loss)

Figure 5.14: Latent coordinates of the "oil flow" dataset retrieved by UKR using Huber's loss function with $\delta = 0.01$. Plot a) corresponds to the 5th intermediate homotopy step with a box constraint $\mathcal{X} = [-4.5; 4.5]^2$, whereas plot b) depicts the final model regularized solely by LOO-CV.

5.1.6. Discussion

In this section, we extended the UKR algorithm by the incorporation of general cost functions. While being conceptually straightforward, this extension is quite fruitful as

[5]Here, we again used the tradeoff point $\delta = 0.01$, yielding a nearly pure L_1 characteristics.

it allows us to tune the method to specific noise models or to enhance its robustness towards outliers. We focused on Huber's loss, which can be utilized in conjunction with LOO-CV, and the ϵ-insensitive loss, for which we introduced an associated practical optimization approach. The latter allows regularizing the UKR manifold by the specification of an error tolerance, which is an eminently suitable way to incorporate additional knowledge about a dataset, in particular the range of noise. If no such knowledge or at least intuition is available, the introduction of an error tolerance (as a new parameter) is not an advantange, and one should rather stick to the automatic parameter-free LOO-CV regularization utilizing either Huber's loss or the standard L_2 loss.

We demonstrated our method on both toy and real data, thoroughly evaluating UKR models based on the standard L_2 loss, Huber's loss, and the ϵ-insensitive loss. LOO-CV regularized UKR models based on Huber's loss yielded significantly better results than those based on the L_2 loss for the case of moderate and high noise levels (Gaussian and Laplacian noise), as shown in the "noisy half circle" experiment and for the "Swiss Roll", as well as for datasets with outliers ("noisy half circle", again). The ϵ-insensitive loss function was applied in our "noisy spiral" experiment, and also in an experiment dealing with subsets from the USPS handwritten digit dataset, where it yielded favorable results as compared to LOO-CV regularized models based on the standard L_2 loss.

Future work may address the utilization of further loss functions. Apart from that, also modelled after Support Vector Machines, one could aim at creating sparse ϵ-insensitive UKR models besides using a finite support density kernel.

5.2. UKR with leave-K-out cross-validation

In this section, we show how to extend the leave-one-out cross-validation method for the regularization of UKR models to a more general leave-K-out cross-validation approach. While LOO-CV has the undisputed advantage of being parameter-free, in some cases it leads to undesirably unsmooth manifolds (see for example the "noisy spiral" experiments in Sec. 4.6.1 and 5.1.4). Of course, UKR already features alternative regularization approaches, for example the usage of penalty terms based on extension or density (4.31,4.33), but the corresponding regularization parameters (e.g. pre-factors) are not very intuitive to select. Therefore, the aim of this UKR extension is to provide an easy-to-handle tool for fitting smooth UKR manifolds. Please note that the concepts and some of the results presented in this section have already been published (Klanke and Ritter, 2006a).

As we have described in section 1.3.4, LOO-CV is a special case of M-fold cross-validation, where generally M subsets of the data are formed, each of which is used to assess the performance of a submodel trained on the union of the remaining subsets. Selecting a larger M induces a higher computational effort, so LOO-CV with $M = N$ usually (but not with UKR) is the most expensive variant of cross-validation. In supervised learning tasks, LOO-CV is known to produce an almost unbiased estimate of the prediction error, but it can exhibit high variance. In contrast, 5-fold or 10-fold cross-validation have lower variance, but may overestimate the prediction error, especially if the sample size is rather small (Hastie et al., 2001). In an extensive simulation of regression learning tasks, Breiman and Spector (1992) found that 5-fold cross-validation

actually yielded better results than LOO-CV, despite having a much lower computational effort. For the unsupervised task of learning principal curves, Hastie and Stuetzle (1989) observed that a repeated application of LOO-CV for selecting the span of local smoothing (cf. Sec. 2.5) leads to unsmooth curves that almost interpolate the data points[6].

Since in M-fold cross-validation $K \approx N/M$ samples are left out from the complete dataset in order to form the different training subsets, it is also called leave-K-out cross-validation, where again the special case $K = 1$ is identical to LOO-CV. Since UKR comes with LOO-CV "for free", it is interesting to investigate if the concept is applicable for $K > 1$. Jonathan, Krzanowski, and McCarthy (2000) pointed out that there are $N!/(M!(K!)^M)$ different ways of dividing the training set into M groups where each K elements are left out, and that "different partitionings may yield very different performance assessments." Therefore, in the following we will first specify a deterministic scheme for selecting a partitioning. After that, we introduce an accompanying regularizer that helps to reduce border effects, and then we demonstrate the resulting UKR leave-K-out cross-validation (LKO-CV) approach in a number of experiments. We conclude this section with a short discussion of the gain of the new approach.

5.2.1. A leave-K-out partitioning scheme

What is a suitable deterministic scheme for applying LKO-CV to UKR, if we wish to achieve smoother models? With the aim to both maximize and equally distribute the effect of omitting each K data vectors on how UKR fits the manifold, we opt to reconstruct each data vector without itself and its $K-1$ nearest neighbors. Concerning this, please recall that the UKR function (4.18) computes a *locally weighted* average of the dataset. Therefore, normally, each data vector is mainly reconstructed from its neighbors. By omitting the immediate neighbors we shift the weight to data vectors farther away, which forces the kernel centers $\mathbf{x}_i$ to huddle closer together and thus leads to a smoother manifold.

Please note that in contrast to standard LKO-CV, this procedure yields N different subsets of size $N-K$, each being responsible for the reconstruction of *one* data vector. A corresponding objective function, which automatically combines the subset models, can be stated as

$$R_{lko}(\mathbf{X}) = \frac{1}{N}\sum_i \|\mathbf{y}_i - \mathbf{f}_i(\mathbf{x}_i;\mathbf{X})\|^2 = \frac{1}{N}\|\mathbf{Y} - \mathbf{YB}_{lko}(\mathbf{X})\|_F^2 \tag{5.18}$$

$$\mathbf{f}_i(\mathbf{x}) = \sum_{m\notin\mathcal{N}_i} \mathbf{y}_m \frac{K(\mathbf{x}-\mathbf{x}_m)}{\sum_{j\notin\mathcal{N}_i} K(\mathbf{x}-\mathbf{x}_j)}, \tag{5.19}$$

where $\mathcal{N}_i$ describes the index set of neighbors excluded for reconstructing $\mathbf{y}_i$.

In principle, we may consider neighborhoods both in latent space and data space, since a good mapping will preserve the topology anyway. However, it is much simpler to regard only the original neighborhood relationships in data space, because these are

[6] Now, the reader might argue that in UKR LOO-CV is repeatedly applied all the time, and that therefore unsmooth curves should have been expected. However, for example in our "noisy spiral" experiments, the UKR curves are quite unsmooth directly after the scale-optimization, that is, after *a single* application of LOO-CV.

fixed. The latent space neighborhoods may change with every training step, and thus have to be recomputed. Furthermore, convergence is not guaranteed anymore, because the latent variables $\mathbf{X}$ might jump between two "optimal" states belonging to different neighborhood structures.

As with LOO-CV, data space neighborhood LKO-CV can be implemented in UKR with nearly no additional cost. All one has to do is zero-ing certain components of the matrix $\mathbf{B}_{lko}$ before normalizing its column sums to 1. In particular, set $b_{ij} = 0$, if $i \in \mathcal{N}_j$, with fixed and precomputed index sets $\mathcal{N}_j$.

At first sight, one might argue that the whole idea seems somehow strange, especially if the UKR model is initialized by a spectral embedding method (e.g. LLE) which takes into account some K' nearest neighbors for constructing the lower dimensional representation. Thus, in a way, UKR with LKO-CV works against its initialization method. On the other hand, this scheme can be viewed as being complementary. Furthermore, our experiments (Sec. 5.2.3) not only show that the idea is sound, but even indicate that selecting $K = K'$ is not a bad choice at all.

5.2.2. How to get smooth borders

As we will show in the next section, a regularization by LKO-CV does work well at the interior of a manifold, but not at its borders. This results naturally from the topology: at the borders of a 1-D manifold (that is, at the ends of a curve) for example, all K neighbors lie in the same direction. Thus, the nearest data points taking part in reconstructing the end points are exceptionally far away. The other way around, the end points do not have to play a crucial role in the reconstruction of other points, and thus the locations of the corresponding latent points are not fixated enough and can move outwards, generating a region of low density inbetween. Therefore, if after training the curve is sampled by evaluating the normal UKR function (4.18), the ends get very wriggly, especially for larger values of K.

To overcome this problem, we propose to employ an additional regularizer that smoothes at the borders without disturbing LKO-CV in regions that are already handled well. Penalizing the extension of latent space (e.g. by using a penalty term of the form $S(\mathbf{X}) = \|\mathbf{X}\|_F^2$) is a bad choice, since this would affect the manifold as a whole and not only the borders. The same argument applies to a density-based penalty term of the form $S(\mathbf{X}) = -\sum_i \log p(\mathbf{x}_i)$, which favors high densities and thus again smoothes the complete manifold. A possible choice, however, is to penalize the *variance* of the density in latent space. For this, we apply the following penalty term:

$$S(\mathbf{X}) = \frac{1}{N} \sum_i (p(\mathbf{x}_i) - \bar{p}(\mathbf{X}))^2 \quad , \quad \bar{p}(\mathbf{X}) = \frac{1}{N} \sum_j p(\mathbf{x}_j). \tag{5.20}$$

The UKR model is thus regularized by two factors: 1) the LKO parameter K determines the overall smoothness and 2) the penalty term $S(\mathbf{X})$, scaled by an appropiate pre-factor λ, ensures that the smoothness is evenly distributed.

Because these regularizers have more or less independent goals, we may hope that the results show considerable robustness towards the choice of λ. Indeed, for a UKR model of a noisy spiral (Fig. 5.16), there was no visual difference between results for $\lambda = 0.001$ and $\lambda = 0.0001$. Only a much smaller value ($\lambda = 10^{-6}$), led to wriggly ends, again.

5.2.3. Experiments

Noisy spiral

As a first example, we fitted a UKR model to the 2-D "noisy spiral" toy dataset, which again contains 300 samples with noise distributed uniformly in the interval $[-0.1; 0.1]$. In order to generate candidate initializations, we utilized the LLE algorithm with neighborhood sizes $K' = 4 \ldots 12$. According to Sec. 4.4.2, we here optimized the scale of these candidates with respect to the LKO-CV error (5.18), where we varied the LKO-parameter from $K = 1$ (LOO-CV) up to $K = 24$. Independently of the choice of K, the LLE neighborhood parameter $K' = 7$ always turned out to yield the best initialization. For fine-tuning the UKR models, we carried out 500 RPROP steps, now minimizing the LKO-CV error (5.18) with respect to the latent variables $\mathbf{X}$.

Fig. 5.15 shows the results for some values of the LKO-CV parameter K as indicated in the plots. Note how the manifold gets smoother for larger K, without suffering from too much bias towards the inner of the spiral. A bit problematic are the manifolds ends, which get quite wriggly for larger K. Note that $K = K' = 7$ yields a visually satisfactory level of smoothness.

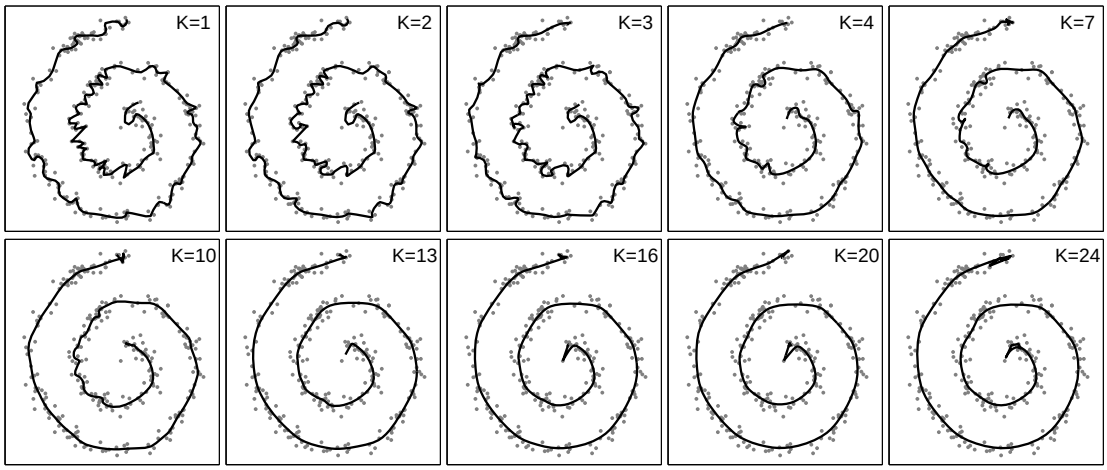


Figure 5.15: UKR model of a noisy spiral using LKO-CV. The data points are depicted as grey dots, and the black curve shows the manifold which results from sampling $\mathbf{f}(\mathbf{x}; \mathbf{X})$.

To show the effect of the density variance penalty term (5.20), we repeated the experiment adding the penalty with pre-factors $\lambda = 10^{-3}, 10^{-4}$ and 10^{-6}. Fig. 5.16 shows the results for $\lambda = 10^{-4}$, which are visually identical to those for $\lambda = 10^{-3}$. However, a pre-factor of only 10^{-6} turned out to be too small, resulting in wriggly ended curves similar to those in Fig. 5.15.

Some insight on the influence of the density variance penalty is provided by Fig. 5.17: Most of the latent variables stay in the same region, but the outliers (the little bumps to the far left and right) are drawn towards the center, compacting the occupied latent space. Figure 5.18 shows a magnified comparison of the UKR models ($K = 24$) with and without the penalty term. In addition to the original data points and the resulting curve, it also depicts the data as it is reconstructed during training, that is, using the leave-K-out function (5.19). Note that these LKO reconstructions show a strong bias

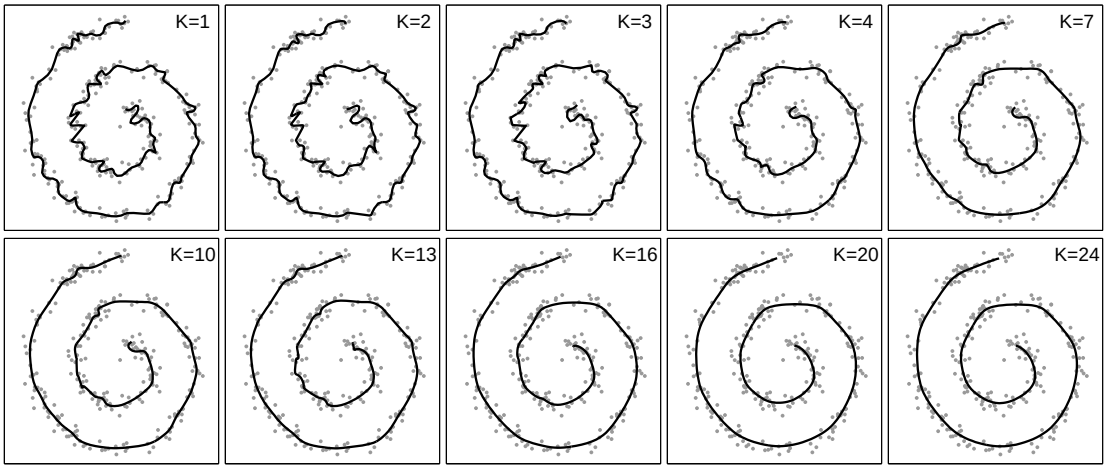


Figure 5.16: UKR model of a noisy spiral using both LKO-CV and the density variance penalty term (5.20) scaled by a pre-factor $\lambda = 10^{-4}$.

towards the inner of the spiral, which in this strength is not present in the final mapping (4.18) based on the complete data set.

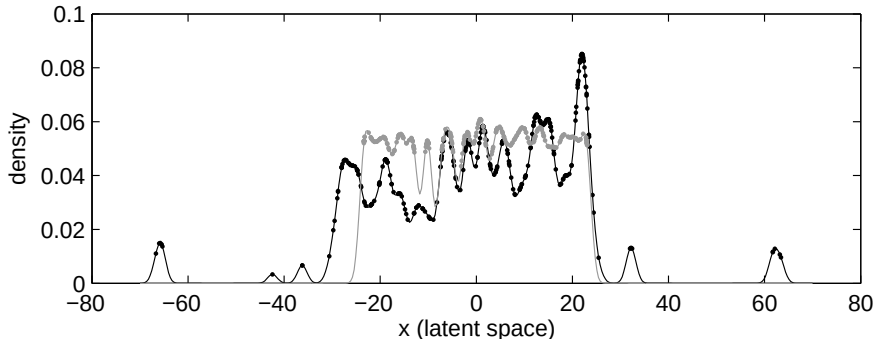


Figure 5.17: Comparison of latent densities for UKR models of a noisy spiral using a) only LKO-CV ($K = 24$, depicted in black) and b) LKO-CV together with the densisty variance penalty ($K = 24$, $\lambda = 10^{-4}$, depicted in gray). The curves result from sampling $p(x)$, the dots indicate the latent variable positions.

For a more thorough evaluation of the influence of LKO-CV, we repeated the experiment for 50 "noisy spiral" datasets all drawn from the same distribution. Actually these datasets were picked from the set of 100 spirals that we already investigated with the ϵ-insensitive loss extension (Sec. 5.1.4), where we made sure that at least one of the LLE candidates yielded a correct unwinding of the spiral. Similar to our reasoning in the experiment with the ϵ-insensitive loss, we carried out the selection because we solely wanted to focus on the smoothing effect of LKO-CV.

For each of the 50 datasets, we trained 20 UKR models utilizing LKO-CV with a LKO parameter $K = 1 \ldots 20$ and a fixed pre-factor $\lambda = 10^{-4}$ for the density variance penalty. Similarly to Sec. 5.1.4, we then measured the mean distance between the UKR reconstructions $\mathbf{f}(\mathbf{x}_i; \mathbf{X})$ and the "true" spiral. Figure 5.19 shows the results averaged across all 50 datasets. Please note the significantly reduced distance resulting for a wide

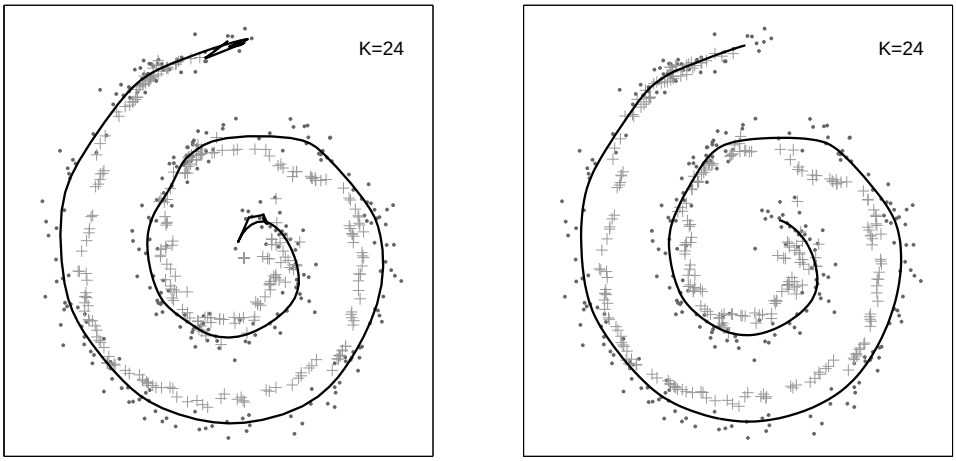


Figure 5.18: Comparison of UKR models of a noisy spiral. Left: pure LKO-CV ($K = 24$). Right: with additional density variance penalty ($\lambda = 10^{-4}$). The dots depict the observed data points, the black curve depicts the manifold, and the gray pluses depict the LKO reconstructions (5.19).

range of LKO parameters ($K = 5 \dots 12$) as compared to the LOO-CV models ($K = 1$).

Next, we investigated a possible relation between the LKO parameter K and the LLE paramter K'. Concerning this, please recall that LLE reconstructs each data vector from a linear combination of its K' nearest neighbors in order to determine corresponding weights for the latent space coordinates (Sec. 2.8.2). Since a linear reconstruction is also involved in the UKR algorithm, albeit with a different meaning of the weights, an empirical investigation of that relation is interesting. To this end, for each of the 50 datasets, we determined the LKO parameter K_{opt} yielding the minimal mean distance to the true spiral, and compared that value to the automatically selected neighborhood size K_{lle} of LLE. A histogram of the difference between these parameters is depicted in Fig. 5.20. While the spread of the differences $K_{opt} - K_{lle}$ is quite wide, in most of the 50 cases the two parameters were close. As a possible explanation, one could argue that if a particular neighborhood size K_{lle} was selected, then that parameter is somehow characteristic for the distribution of the data, and as such might be a good choice for the leave-K-out method, also. As an argument against using a corresponding heuristics $K = K'$, different spectral methods do not necessarily work best for the same neighborhood setting[7], so the degree of smoothing would depend on the choice of the spectral method, and not on the data alone. Fortunately, as can be seen from Fig. 5.19, a wide range of leave-K-out parameters K yields comparably good results, so even a coarse heuristics for choosing the parameter is useful.

As a last part of our experiments with the "noisy spiral" dataset, we compared the LKO-CV regularization method directly to our proposition of smoothing UKR manifolds via the ϵ-insensitive loss (Sec. 5.1.4). In particular, we utilized the same 95 "noisy spiral" datasets on which we collected the statistics of the ϵ-insensitive UKR curves, together with the same initialization candidates as provided by Isomap. We then carried out up to 1000 RPROP steps minimizing the sum of the LKO-CV error (5.18) and the density

[7] We noted that Isomap typically requires a smaller neighborhood parameter than LLE.

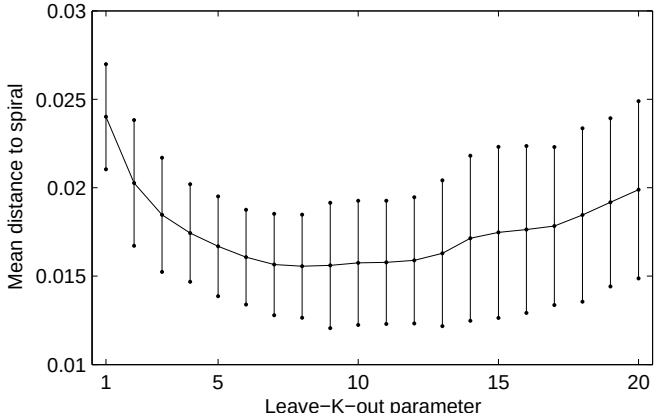


Figure 5.19: Mean distance between UKR reconstructions of the "noisy spiral" data and the underlying smooth manifold as a function of the leave-K-out parameter K. The results are averaged across 50 datasets,with the bars indicating one standard deviation.

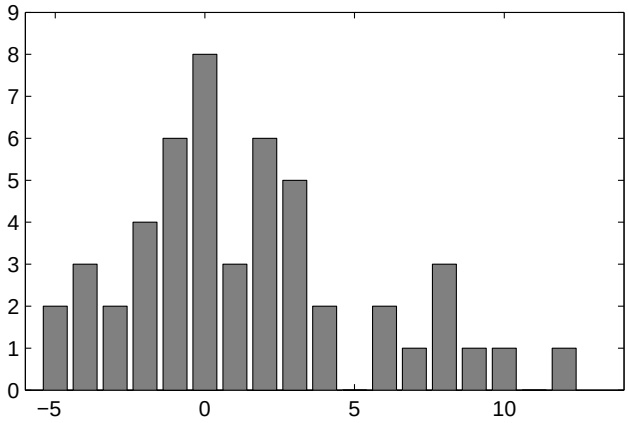
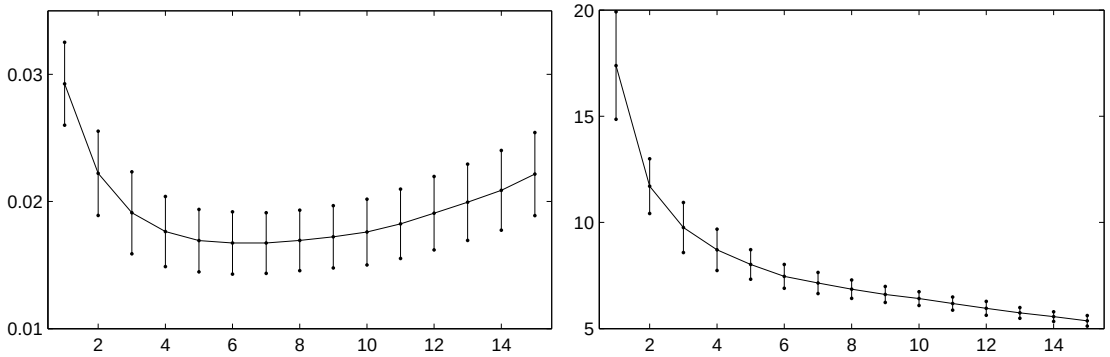


Figure 5.20: Histogram of the quantity $K_{opt} - K_{lle}$, where K_{opt} denotes the LKO parameter yielding the smallest distance to the underlying spiral, and K_{lle} denotes the neighborhood parameter of LLE that led to the best candidate initialization.

variance penalty (5.20), again scaled by a pre-factor $\lambda = 10^{-4}$.

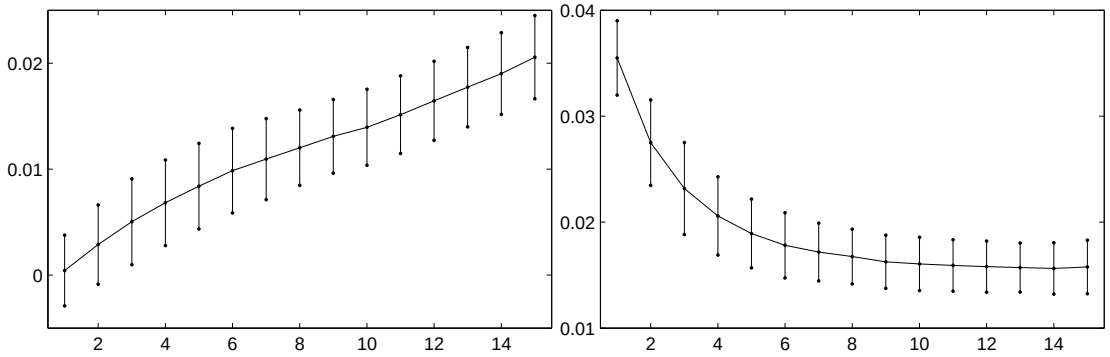
Just as in the corresponding experiment with the ϵ-insensitive loss, we evaluated the mean distance of the sampled UKR manifold to the underlying spiral, as well as the extension of latent space, the mean *signed* distances (5.16), and the standard deviation of these distances as a measure of roughness. The results are depicted in Fig. 5.21. Overall, the performance of the UKR models based on LKO-CV and the ϵ-insensitive loss (Fig. 5.8 and 5.9) is very similar, with a slight advantage of the LKO-CV models in terms of the mean distance (Fig. 5.21a and 5.8).

Regarding conceptual differences, the optimization scheme of UKR models utilizing LKO-CV is simpler, but the ϵ-insensitive UKR models have the advantage that the regularization parameter ϵ might already be pre-defined by the application.



a) mean distance between UKR manifold and underlying true spiral.

b) extension of latent space as given by $\|\mathbf{X}\|_F$.



c) mean *signed* distance $\bar{\Delta}$, higher values indicate a bias towards the inner of the spiral.

d) standard deviation of signed distances as a roughness measure of the UKR manifold.

Figure 5.21: LKO-CV regularized UKR models of "noisy spiral" datasets. The above plots show the same statistics as Fig. 5.8 and 5.9, but here the horizontal axis corresponds to the LKO parameter K. Results are averaged across 95 datasets, with the bars depicting one standard deviation.

USPS digits

To show that UKR with LKO-CV can also be applied to higher dimensional data, our last experiment of this section deals with the USPS handwritten digits. In particular, for visualization purposes we first worked with the subset corresponding to the digit "2" (731 data vectors in 256 dimensions). We initialized our UKR models with LLE (cf. Sec. 4.6.3), and compared the results of LOO-CV and LKO-CV with $K = K' = 12$, that is, we chose the LKO parameter to be identical to the automatically selected LLE neighborhood size. Both models use the density variance penalty with a pre-factor[8] $\lambda = 0.01$. Figure 5.22 visualizes the resulting manifolds (we chose a 2-D embedding) by sampling $\mathbf{f}(\mathbf{x}; \mathbf{X})$ in latent space and depicting the function value as the corresponding image. Note the smaller extension in latent space and the blurrier images of the model belonging to $K = 12$ (right plot).

In a follow-up experiment, we again built a digit classifier in the way of section 4.6.3, using the same homotopy-based optimization approach and the same choices of latent space dimensionality and Mercer kernel. In contrast to Sec. 4.6.3, however, we

[8]Here, we used a larger λ because the data's variance is larger, too.

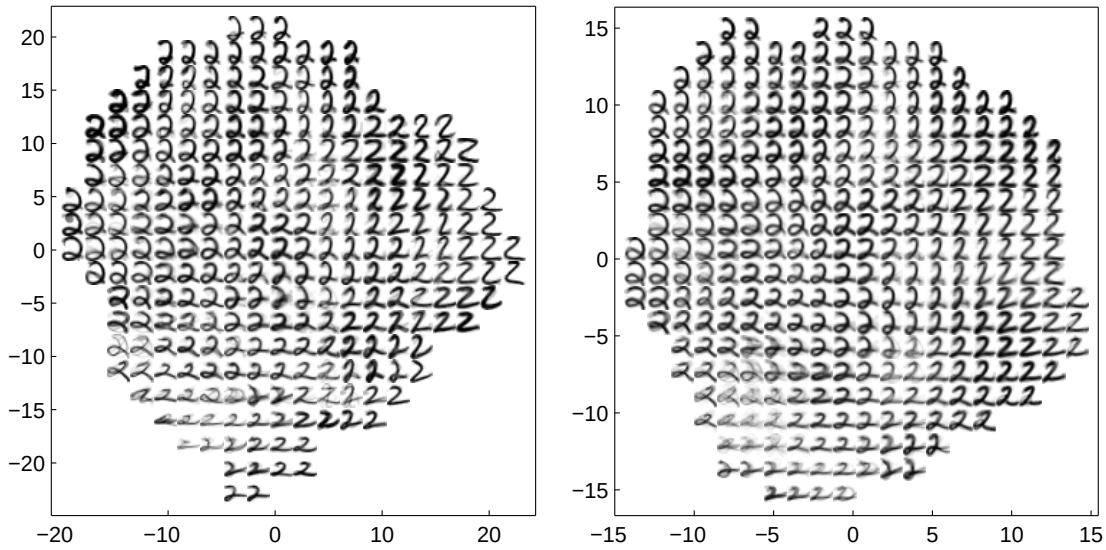


Figure 5.22: UKR model of the USPS digit "2", shown by evaluating $\mathbf{f}(\mathbf{x}; \mathbf{X})$ on a 20x20 grid enclosing the latent variables. Grid positions of low density $p(\mathbf{x})$ are left blank. Left: $K = 1$ (LOO-CV). Right: $K = 12$.

here regularized the model with LKO-CV, using small or moderate parameter values $K = 2 \ldots 5$, and we also included the density variance penalty with a pre-factor $\lambda = 0.01$. Table 5.7 summarizes the resulting error rates. While in some cases (e.g. data space UKR, $q = 9$) utilizing LKO-CV increased the error rate, especially for the lower dimensional models the classification performance was improved.

Method	$q = 5$	$q = 6$	$q = 7$	$q = 8$	$q = 9$	$q = 10$	$q = 11$	$q = 12$
Data Space UKR, LOO-CV	4.78	4.78	4.53	4.53	4.19	**3.94**	3.99	4.19
LKO-CV, K=2	4.04	4.14	3.99	4.63	4.83	4.33	4.33	**3.94**
LKO-CV, K=3	4.14	4.24	4.19	4.24	4.68	4.43	4.24	4.33
LKO-CV, K=4	3.99	**3.94**	4.24	4.43	4.88	4.19	4.04	4.14
LKO-CV, K=5	4.04	**3.94**	3.99	4.43	4.73	3.99	4.09	4.24
Polynomial Kernel, LOO-CV	4.43	4.58	4.19	4.24	4.14	4.29	4.24	4.33
LKO-CV, K=2	4.33	4.19	4.19	4.68	4.68	4.19	4.24	4.24
LKO-CV, K=3	4.38	4.24	4.24	4.53	5.08	4.38	4.29	4.19
LKO-CV, K=4	4.38	**4.04**	4.24	4.24	4.48	4.24	**4.04**	4.33
LKO-CV, K=5	4.19	4.33	4.19	4.43	4.29	4.19	4.14	4.29

Table 5.7: Error rates (in percent) of the UKR digit classifier for different dimensions of the latent space and different choices of the leave-K-out parameter. The lowest error rates for data space UKR models and models based on the polynomial kernel are printed in bold face.

5.2.4. Discussion

In this section, we described how leave-K-out cross-validation (LKO-CV) can be employed in the manifold learning method UKR, generalizing the already present LOO-CV

regularization. We demonstrated our approach on both synthetic and real data. When used with pre-calculated data space neighborhoods, LKO-CV involves nearly no additional computational cost, but can yield favorable results. This was revealed especially in the "noisy data" experiment, where LKO-CV significantly reduced the projection error, i.e. the mean distance between the reconstructed (de-noised) dataset and the "true" underlying manifold.

While we gave no final answer to the question how to choose the new regularization parameter K, our experiments indicate that simply setting $K = K'$ (the neighborhood size of the best LLE solution, which UKR can automatically detect) yields satisfactory results. If other methods for initialization and optimization are applied, one has little guidance for the selection of K, but even in that case LKO-CV with a "small" ad-hoc choice for K is worth considering (cf. the digit classification performance). A costlier, but theoretically more satisfying approach for selecting K would be to compare the results of different choices with regard to the projection error of hold-out data (cf. Sec. 1.3.4).

In this section, we also showed how a complementary regularizer, which is based on penalizing a high variance of the latent density, can further enhance the UKR models trained with LKO-CV. By promoting an even distribution of smoothness, this regularizer diminishes the problem of rather wriggly manifold borders, which otherwise may result from a pure LKO-CV regularization. When used as a penalty term, the complementary regularizer is quite robust towards the choice of an appropiate pre-factor.

Further work may address other possibilities to deal with the border problem, e.g. by a smart local adaption of the neighborhood parameter K. We also succesfully experimented with leave-R-out CV, a scheme where not a fixed number of neighbors are left out, but all neighbors within a sphere of fixed size.

5.3. Landmark UKR

In this section, we will present a variant of UKR which is mainly aimed at reducing the computational effort. The concepts presented here have not yet been published, and they have also not been investigated and evaluated as thoroughly as the two preceding UKR extensions. Therefore, the nature of this section is rather prospective, although experiments conducted as a proof of concept have shown promising results.

As has been discussed in section 4.4.1, the complexity of a single evaluation of the UKR reconstruction error or its gradient is in the class $O(dN^2)$, that is, it is quadratic in the number of training samples. In case of feature space UKR (Sec. 4.5) the complexity is even $O(N^3)$, which quickly becomes impractical for larger training sets. In chapter 2, we have seen several methods that suffer from the same problem, most notably nonlinear spectral embedding methods (Sec. 2.8) with an eigendecomposition of an $N \times N$ matrix, and also iterative MDS algorithms like Curvilinear Component Analysis (CCA, Sec. 2.7.3) and Curvilinear Distance Analysis (CDA, Sec. 2.7.4). In order to address that problem, "landmark" variants of spectral methods have been designed by de Silva and Tenenbaum (2004) and Weinberger et al. (2005). Moreover, both CCA and CDA effectively only operate on a "reduced" dataset as produced by a preceding vector quantization step. These ideas can be carried over to UKR, as we will do here.

The remainder of this section is organized as follows: firstly, we will present the concept of "landmark UKR" together with a corresponding modification of the UKR objective function. Then, we will motivate a further post-processing training step, for which we carry the explicit roughness measure of the PSOM$^+$ (Sec. 3.3.1) over to UKR. After that, we will demonstrate the new variant experimentally. The section will be concluded by a discussion of the gains as well as of open problems and future research directions.

5.3.1. Reconstruction from landmark points

The prime motivation for the UKR variant described here is to reduce the computational effort that is involved in both training the model and using it afterwards, i.e. for projecting new data and sampling from the manifold. While already the application of finite support density kernels (e.g. Quartic) can lead to a much reduced computational burden, the improvement in efficiency still leaves much to be desired, first of all in situations where the homotopy-based optimization scheme has to be applied. Furthermore, also when a UKR manifold is smoothed, for example by using the ϵ-insensitive loss function (Sec. 5.1) or by training with leave-K-out cross-validation (Sec. 5.2), the efficiency gains of finite support kernels are diminished due to the stronger overlapping of the basis functions. This connection is not without irony, because smoother manifolds usually coincide with simpler functions, and therefore should be *cheaper* to compute.

An early attempt to create efficient UKR models similar in spirit to support vector machines has not been successful. By adding extra weights to the UKR function model, for example in the way of

$$\mathbf{f}(\mathbf{x};\mathbf{X};\mathbf{w}) = \sum_i \mathbf{y}_i \frac{w_i k(\mathbf{x}-\mathbf{x}_i)}{\sum_j w_j k(\mathbf{x}-\mathbf{x}_j)} \quad \text{with} \quad \forall_i\ w_i \geq 0, \tag{5.21}$$

one can enforce sparseness by penalizing non-zero weights when minimizing the reconstruction error. However, in practice, it turned out to be hard to find a good optimization scheme for adapting both the latent variables and the weights. Furthermore, the computational effort is not reduced right from the start of the training process, but in fact increased by adding extra parameters.

Please note that the aforementioned sparsification attempt basically implies a partitioning of the training data $\mathbf{Y}$ into a set of vectors supporting the model (vectors $\mathbf{y}_i$ corresponding to non-zero weights w_i) and another "validation" set of vectors whose weights are zero. This creates some asymmetry, as parts of the data are handled in a different manner. If one takes a supporting vector $\mathbf{y}_*$ and projects it onto the final UKR model, the resulting position in latent space will likely be *not* identical to $\mathbf{x}_*$ as determined from the training: during projection, a data vector $\mathbf{y}$ seeks the position $\mathbf{x}$ that is most fit only for itself, while during training a supporting data vector is also responsible for reconstructing others.

Therefore, we follow an alternative approach and first *extend* the training data $\mathbf{Y}$ by adding $n \ll N$ support vectors or landmark points $\hat{\mathbf{Y}} = (\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2 \ldots \hat{\mathbf{y}}_n)$, which are retrieved by some clustering or vector quantization algorithm. In our experiments, we chose the K-Harmonic Means algorithm (Sec. 2.1.3), since it is parameter-free (apart from the number of cluster centers) and has been shown to be virtually insensitive

towards bad initializations[9]. Then, we reconstruct $\mathbf{Y}$ from a UKR model utilizing only $\hat{\mathbf{Y}}$ in the modified regression function

$$\mathbf{f}(\mathbf{x}; \hat{\mathbf{X}}) = \hat{\mathbf{Y}}\mathbf{b}(\mathbf{x}; \hat{\mathbf{X}}), \tag{5.22}$$

where $\hat{\mathbf{X}} = (\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2 \dots \hat{\mathbf{x}}_n)$ denotes the latent space representations of the landmark points. As our new training goal, we minimize the reconstruction error

$$R(\mathbf{X}, \hat{\mathbf{X}}) = \frac{1}{N} \sum_i \|\mathbf{y}_i - \hat{\mathbf{Y}}\mathbf{b}(\mathbf{x}_i; \hat{\mathbf{X}})\|^2 = \frac{1}{N} \|\mathbf{Y} - \hat{\mathbf{Y}}\mathbf{B}(\mathbf{X}, \hat{\mathbf{X}})\|_F^2, \tag{5.23}$$

where the matrix of basis functions $\mathbf{B}(\mathbf{X}, \hat{\mathbf{X}})$ is given by

$$(\mathbf{B}(\mathbf{X}, \hat{\mathbf{X}}))_{ij} = \frac{K(\hat{\mathbf{x}}_i - \mathbf{x}_j)}{\sum_k K(\hat{\mathbf{x}}_k - \mathbf{x}_j)}. \tag{5.24}$$

Please note that we minimize (5.23) with respect to both $\mathbf{X}$ and $\hat{\mathbf{X}}$, that is, we seek optimal latent coordinates for both the original data points and for the generated landmark points. As with standard UKR, these latent coordinates are the only parameters of the model, besides the choice of the density kernel $K(\cdot)$ and – added in this variant of UKR – the number n of landmark points, which has to be selected in advance.

Please also note that if we fix $\hat{\mathbf{X}}$, we can treat $(\hat{\mathbf{Y}}, \hat{\mathbf{X}})$ as data vectors and latent variables of a readily trained standard UKR model. Therefore, the reconstruction error (5.23) just represents the mean projection error for new data $\mathbf{Y}$, and the aforementioned asymmetry is removed. The gradients $\frac{\partial R}{\partial \mathbf{X}}$ and $\frac{\partial R}{\partial \hat{\mathbf{X}}}$ have a remarkable symmetry and can be expressed as

$$\frac{\partial R}{\partial \mathbf{X}} = \frac{2}{N} \left[\hat{\mathbf{X}}\mathbf{Q} - \mathbf{X} \cdot \operatorname{diag}\left(\mathbf{1}_n^T \mathbf{Q}\right) \right] \tag{5.25}$$

$$\frac{\partial R}{\partial \hat{\mathbf{X}}} = \frac{2}{N} \left[\mathbf{X}\mathbf{Q}^T - \hat{\mathbf{X}} \cdot \operatorname{diag}\left(\mathbf{1}_N^T \mathbf{Q}^T\right) \right] \tag{5.26}$$

$$\mathbf{Q} = \mathbf{P} * \left[\mathbf{M} - \mathbf{1}_n \mathbf{1}_N^T (\mathbf{B} * \mathbf{M}) \right] \tag{5.27}$$

$$\mathbf{M} = \hat{\mathbf{Y}}^T (\hat{\mathbf{Y}}\mathbf{B} - \mathbf{Y}) \tag{5.28}$$

Here, the matrix $\mathbf{P}$ is defined analogous to the formula (4.44) of standard UKR. For this variant, however, $\mathbf{Q}, \mathbf{P}, \mathbf{B}$, and $\mathbf{M}$ are $n \times N$ matrices, in contrast to $N \times N$ as in a non-landmark UKR model. Consequently, the computational effort of an evaluation of (5.23) and its gradients is reduced to $O(dnN)$, where still the most time is spent for calculating the matrix $\mathbf{M}$. Please refer to appendix C.3 for a detailed derivation.

Feature space variant

The matrix $\mathbf{M}$ can be expressed by dot products between data vectors, and therefore the "kernel trick" (Sec. 1.5.4) can be applied to landmark UKR, too:

$$\mathbf{M} = \hat{\mathbf{Y}}^T \hat{\mathbf{Y}}\mathbf{B} - \hat{\mathbf{Y}}^T \mathbf{Y} = \mathbf{\Psi}\mathbf{B} - \mathbf{\Phi}. \tag{5.29}$$

[9] In a similar manner, the K-Harmonic Means algorithm has also been combined with the Generative Topographic Mapping (Sec. 2.6.1) by Peña and Fyfe (2006).

Here, $\mathbf{\Psi}$ ("kernel matrix of landmark points") is an $n \times n$ matrix given by $\psi_{ij} = k(\hat{\mathbf{y}}^i, \hat{\mathbf{y}}^j)$ and $\mathbf{\Phi}$ ("mixed kernel matrix") is an $n \times N$ matrix defined by $\phi_{ij} = k(\hat{\mathbf{y}}^i, \mathbf{y}^j)$. If the training set is not given in a traditional vector space form, but only as a Mercer kernel matrix (e.g. for structured data), one can utilize a kernelized K-Means algorithm (Schölkopf et al., 1998) for generating feature space landmark points.

Note that the computational effort is now $O(n^2 N)$, instead of $O(N^3)$ for the feature space variant of standard UKR.

Initialization

In order to initialize a landmark UKR model, nonlinear spectral embedding methods can be utilized just as with standard UKR. Note, however, that now $N + n$ vectors (original data and landmark points) have to be embedded, yielding a slight increase of the computational effort. A natural and efficient way to provide initial low dimensional coordinates $(\mathbf{X}, \hat{\mathbf{X}})$ of the combined set $(\mathbf{Y}, \hat{\mathbf{Y}})$ is to also apply a landmark variant of the spectral method, e.g. landmark Isomap (Sec. 2.8.1) or ℓSDE (Sec. 2.8.3).

Regularization

If N data vectors are reconstructed from a convex combination of $n \ll N$ landmark points, in general the data vectors cannot be trivially interpolated. Nonetheless, one should include a regularization term, e.g. a density or extension-based penalty, in order to prevent a too wide distribution of the latent variables $\mathbf{X}$ and $\hat{\mathbf{X}}$. Otherwise, if the kernel functions centered at the locations $\hat{\mathbf{x}}_i$ have only little or no overlap, the UKR manifold will interpolate the landmark points with sharp transitions inbetween. Up to now, we have no good heuristic or even an automated choice of the penalty term and its pre-factor.

5.3.2. Landmark adaption and smoothness control

We have already discussed in Sec. 5.1.4 that the UKR function (4.18) is inherently biased in the presence of curvature, since it is a convex combination of data vectors. We have also seen that this bias (e.g. towards the inner of a spiral) gets stronger for smoother manifolds. The same reasoning applies to the landmark variant of the UKR function (5.22).

In effect, the new parameter n of landmark UKR is not only responsible for controlling the computational complexity, but also the shape complexity of the manifold. If the landmark points are generated by a vector quantization (VQ) pre-processing step, they already are the product of local averaging, and therefore the landmark UKR function can be viewed as a *double* local average. A smaller number n of landmark points implies a wider span of the averaging within the VQ step, and thus we should not be surprised to observe an increasing bias.

To overcome this problem, we propose to view also the landmark points $\hat{\mathbf{Y}}$ as parameters of the UKR model, which therefore can be modified as part of the training. Since the model is linear in $\hat{\mathbf{Y}}$, optimizing these parameters will be much more efficient than optimizing the latent coordinates: for fixed latent variables $(\mathbf{X}, \hat{\mathbf{X}})$, a minimization of

the reconstruction error (5.23) with respect to $\hat{\mathbf{Y}}$ requires finding a least squares solution of the overdetermined system $\mathbf{Y} = \hat{\mathbf{Y}}\mathbf{B}(\mathbf{X}, \hat{\mathbf{X}})$.

However, care must be taken that the extra flexibility does not lead to overfitting, which may result when utilizing and adapting a rather large number of landmark points. Therefore, one should include a regularization term in the landmark adaption step.

In the remainder of this section, we introduce a smoothness metric in the space of the new UKR parameters (or "weights") $\hat{\mathbf{Y}}$. We use the same formalism as for the PSOM$^+$ (Sec. 3.3.1), that is, we integrate the square-sum of all second derivatives, treating each data space dimension separately. This yields the following measure of roughness[10] of the landmark UKR manifold:

$$\begin{aligned}
\tilde{R}(\hat{\mathbf{X}}, \hat{\mathbf{Y}}) &= \sum_{m=1}^{d} \int_{\mathcal{X}} \sum_{\mu=1}^{q} \sum_{\nu=1}^{q} \left(\frac{\partial}{\partial x^\mu} \frac{\partial}{\partial x^\nu} f_m(\mathbf{x}; \hat{\mathbf{X}}) \right)^2 d^q x \\
&= \sum_{m=1}^{d} \int_{\mathcal{X}} \sum_{\mu,\nu=1}^{q} \left(\frac{\partial}{\partial x^\mu} \frac{\partial}{\partial x^\nu} \left(\sum_{i=1}^{n} \hat{y}_{mi} b_i(\mathbf{x}; \hat{\mathbf{X}}) \right) \right)^2 d^q x \\
&= \sum_{m=1}^{d} \sum_{i,j=1}^{n} \sum_{\mu,\nu=1}^{q} \hat{y}_{mi} \hat{y}_{mj} \underbrace{\int_{\mathcal{X}} \left(\frac{\partial^2 b_i(\mathbf{x}; \hat{\mathbf{X}})}{\partial x^\mu \partial x^\nu} \right) \left(\frac{\partial^2 b_j(\mathbf{x}; \hat{\mathbf{X}})}{\partial x^\mu \partial x^\nu} \right) d^q x}_{\mathcal{I}_{ij}^{\mu\nu}(\hat{\mathbf{X}})} \\
&= \sum_{m=1}^{d} \sum_{\mu,\nu=1}^{q} \sum_{i,j=1}^{n} \hat{y}_{mi} \hat{y}_{mj} \mathcal{I}_{ij}^{\mu\nu}(\hat{\mathbf{X}}) \\
&= \operatorname{tr}\left(\hat{\mathbf{Y}} \tilde{\mathbf{M}} \hat{\mathbf{Y}}^T \right) \qquad (5.30)
\end{aligned}$$

where $(\tilde{\mathbf{M}})_{ij} = \sum_{\mu,\nu} \mathcal{I}_{ij}^{\mu\nu}$. With a fixed choice of the density kernel, the integrals $\mathcal{I}_{ij}^{\mu\nu}$ solely depend on $\hat{\mathbf{X}}$ – similar to the PSOM$^+$, where the integrals solely depend on the spacing of the nodes. For the PSOM$^+$ algorithm, the integrals $\mathcal{I}_{ij}^{\mu\nu}$ could be solved analytically, but here, given the more complex basis functions $\mathbf{b}(\mathbf{x}, \hat{\mathbf{X}})$, this is in general not possible. Moreover, the region $\mathcal{X}$ we integrate over is not a simple hyper-rectangle anymore, complicating the integration even further.

A practical solution to this problem is to approximate the integral by a discrete sum and thus to sample the "roughness" in latent space, for example at all locations $\mathbf{x}_j,\ j = 1 \ldots N$, that is, at the locations where the training data is projected to. The computational effort to calculate the metric $\tilde{\mathbf{M}} \in \mathbb{R}^{n \times n}$ is proportional to the number of sampling positions, but goes quadratically with the number of landmark points. If $\tilde{\mathbf{M}}$ is only used to optimize the weights $\hat{\mathbf{Y}}$ after optimizing the $\hat{\mathbf{X}}$, it has to be calculated only once, so this is not a problem.

Since the roughness measure (5.30) involves second derivatives of the basis functions, the Quartic kernel can not be utilized here. As a possible replacement, the Triweight density kernel

$$K(\mathbf{x} - \mathbf{x}') \propto \left[1 - \|\mathbf{x} - \mathbf{x}'\|^2 \right]_+^3 \qquad (5.31)$$

[10] To avoid a clash of notation, we here indicate all quantities related to roughness by a tilde, e.g. $\tilde{R}$.

also has finite support, but continuous second derivatives. In the following we present the necessary expressions for calculating the roughness measure. To this end, we introduce two abbreviations $b_i^\mu(\mathbf{x})$ and $b_i^{\mu\nu}(\mathbf{x})$ that represent a normalized first and second derivative of the i-th basis function. All summations are to be understood to range from 1 to n.

$$\begin{aligned}
\frac{\partial}{\partial x^\mu} b_i(\mathbf{x}) &= \frac{\partial}{\partial x^\mu} \frac{K(\mathbf{x}-\hat{\mathbf{x}}_i)}{\sum_j K(\mathbf{x}-\hat{\mathbf{x}}_j)} \\
&= \underbrace{\frac{\frac{\partial}{\partial x^\mu} K(\mathbf{x}-\hat{\mathbf{x}}_i)}{\sum_j K(\mathbf{x}-\hat{\mathbf{x}}_j)}}_{b_i^\mu(\mathbf{x})} - \frac{K(\mathbf{x}-\hat{\mathbf{x}}_i)}{\left(\sum_l K(\mathbf{x}-\hat{\mathbf{x}}_l)\right)^2} \sum_j \frac{\partial}{\partial x^\mu} K(\mathbf{x}-\hat{\mathbf{x}}_j) \\
&= b_i^\mu(\mathbf{x}) - b_i(\mathbf{x}) \sum_j b_j^\mu(\mathbf{x}) \qquad (5.32) \\
\frac{\partial}{\partial x^\nu} b_i^\mu(\mathbf{x}) &= \underbrace{\frac{\frac{\partial}{\partial x^\nu}\frac{\partial}{\partial x^\mu} K(\mathbf{x}-\hat{\mathbf{x}}_i)}{\sum_j K(\mathbf{x}-\hat{\mathbf{x}}_j)}}_{b_i^{\mu\nu}(\mathbf{x})} - \frac{\frac{\partial}{\partial x^\mu} K(\mathbf{x}-\hat{\mathbf{x}}_i)}{\left(\sum_l K(\mathbf{x}-\hat{\mathbf{x}}_l)\right)^2} \sum_j \frac{\partial}{\partial x^\nu} K(\mathbf{x}-\hat{\mathbf{x}}_j) \\
&= b_i^{\mu\nu}(\mathbf{x}) - b_i^\mu(\mathbf{x}) \sum_j b_j^\nu(\mathbf{x}) \qquad (5.33)
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial x^\nu}\frac{\partial}{\partial x^\mu} b_i(\mathbf{x}) &= b_i^{\mu\nu}(\mathbf{x}) - b_i^\mu(\mathbf{x}) \sum_j b_j^\nu(\mathbf{x}) \\
&\quad - \left(b_i^\nu(\mathbf{x}) - b_i(\mathbf{x}) \sum_m b_m^\nu(\mathbf{x}) \right) \sum_j b_j^\mu(\mathbf{x}) \\
&\quad - b_i(\mathbf{x}) \sum_j \left(b_j^{\mu\nu}(\mathbf{x}) - b_j^\mu(\mathbf{x}) \sum_m b_m^\nu(\mathbf{x}) \right) \\
&= b_i^{\mu\nu}(\mathbf{x}) - b_i(\mathbf{x}) \sum_j b_j^{\mu\nu}(\mathbf{x}) \\
&\quad - b_i^\mu(\mathbf{x}) \sum_j b_j^\nu(\mathbf{x}) - b_i^\nu(\mathbf{x}) \sum_j b_j^\mu(\mathbf{x}) \\
&\quad + 2 b_i(\mathbf{x}) \sum_j b_j^\mu(\mathbf{x}) \sum_m b_m^\nu(\mathbf{x}) \qquad (5.34)
\end{aligned}$$

Again, we restrict ourselves to the case of spherically symmetric kernels, that is

$$\begin{aligned}
K(\mathbf{x}-\hat{\mathbf{x}}) &= F(\|\mathbf{x}-\hat{\mathbf{x}}\|^2) \qquad (5.35) \\
\frac{\partial}{\partial x^\mu} K(\mathbf{x}-\hat{\mathbf{x}}) &= 2F'(\|\mathbf{x}-\hat{\mathbf{x}}\|^2)(x_\mu - \hat{x}_\mu) \qquad (5.36) \\
\frac{\partial}{\partial x^\mu}\frac{\partial}{\partial x^\nu} K(\mathbf{x}-\hat{\mathbf{x}}) &= 4F''(\|\mathbf{x}-\hat{\mathbf{x}}\|^2)(x_\mu - \hat{x}_\mu)(x_\nu - \hat{x}_\nu) \\
&\quad + 2\delta_{\mu\nu} F'(\|\mathbf{x}-\hat{\mathbf{x}}\|^2). \qquad (5.37)
\end{aligned}$$

Given the metric $\tilde{\mathbf{M}}$, we can smoothly adapt a readily trained landmark UKR manifold just as with the PSOM$^+$: We minimize the reconstruction error with respect to the

"weights" $\hat{\mathbf{Y}}$, but add the roughness term (5.30) as a penalty. This yields a term which is quadratic in $\hat{\mathbf{Y}}$, so the minimum can be retrieved by setting the gradient to zero.

$$R_\gamma(\hat{\mathbf{Y}}) = \|\mathbf{Y} - \hat{\mathbf{Y}}\mathbf{B}(\mathbf{X}, \hat{\mathbf{X}})\|_F^2 + \gamma \cdot \mathrm{tr}\left(\hat{\mathbf{Y}}\tilde{\mathbf{M}}\hat{\mathbf{Y}}^T\right) \tag{5.38}$$

$$0 \stackrel{!}{=} \frac{\partial R_\gamma}{\partial \hat{\mathbf{Y}}} = 2(\mathbf{Y} - \hat{\mathbf{Y}}\mathbf{B})(-\mathbf{B}^T) + 2\gamma\hat{\mathbf{Y}}\tilde{\mathbf{M}} \tag{5.39}$$

$$\Rightarrow \hat{\mathbf{Y}}_{opt} = \mathbf{Y}\mathbf{B}^T(\gamma\tilde{\mathbf{M}} + \mathbf{B}\mathbf{B}^T)^{-1} \tag{5.40}$$

Just as with the PSOM$^+$, the parameter γ balances between a smoother manifold and a closer fit of the data vectors. Please note that these formulas also work in the case of non-landmark UKR models, in which case one sets $\mathbf{X} = \hat{\mathbf{X}}$ and also "duplicates" the data matrix $\mathbf{Y}$ to get a weight matrix $\hat{\mathbf{Y}}$.

5.3.3. Experiments

Noisy Spiral

As a first experiment, we fitted landmark UKR models to a "noisy spiral" dataset containing 500 samples and Gaussian noise with standard deviation $\sigma = 0.05$. For simplicity, we chose the Gaussian density kernel. We generated $n = 30$, $n = 50$ and $n = 100$ landmark points $\hat{\mathbf{Y}}$ by applying the K-Harmonic Means algorithm (Sec. 2.1.3) and calculated initial embeddings of the resulting extended datasets $(\mathbf{Y}, \hat{\mathbf{Y}})$ with 530, 550, and 600 samples by use of the Isomap algorithm. As already described earlier, we compared the results of several settings of the Isomap neighborhood parameter with respect to the reconstruction error (5.23).

Starting from the best scale-optimized candidate, we minimized the reconstruction error plus an extension penalty term (cf. Eq. 4.31)

$$R_\lambda(\mathbf{X}, \hat{\mathbf{X}}) = \frac{1}{N}\|\mathbf{Y} - \hat{\mathbf{Y}}\mathbf{B}(\mathbf{X}, \hat{\mathbf{X}})\|_F^2 + \lambda\left(\|\mathbf{X}\|_F^2 + \|\hat{\mathbf{X}}\|_F^2\right) \tag{5.41}$$

with a pre-factor 0.001 by carrying out 1000 RPROP steps. The pre-factor was chosen experimentally, where the goal was only to reach a sufficient overlapping of the density kernels. The resulting coarse UKR models are depicted in the top row of Fig. 5.23. Note that the manifolds are smooth, but show a considerable bias towards the inner of the spiral.

Next, we fixed the latent variables $(\mathbf{X}, \hat{\mathbf{X}})$, calculated the roughness metric $\tilde{\mathbf{M}}$ by sampling at the locations $\mathbf{X}$, and adapted the landmark points $\hat{\mathbf{Y}}$ as described by (5.40), using three different settings for the pre-factor: $\gamma = 1$, $\gamma = 0.1$ and $\gamma = 0$. Please note that the last case corresponds to an unconstrained minimization of the reconstruction error (5.23) with respect to the landmark points. The resulting manifolds are shown in rows 2 – 4 of Fig. 5.23. For $\gamma = 1$, all three models (differing in the number of landmark points) already are improved, but still exhibit a curvature bias. For $\gamma = 0.1$, the curves are still smooth, but the bias is removed (judged visually), while for $\gamma = 0$ the models with a larger number of landmark points show a wriggly behavior.

Regarding the computational effort, 100 gradient evaluations of our UKR model with 50 landmark points and 500 data points took only 2 seconds, whereas for 100 gradient evaluations of a corresponding standard UKR model 20 seconds were required.

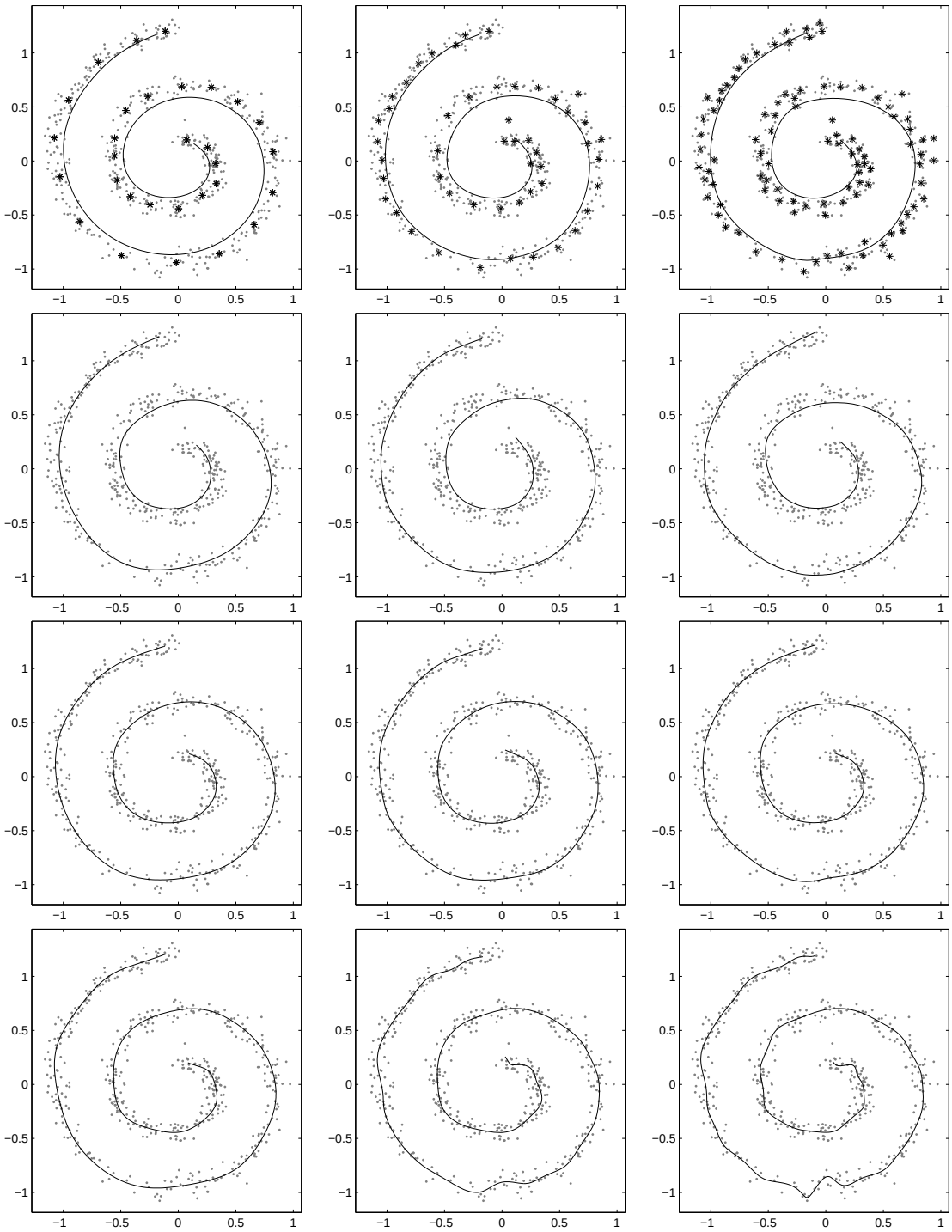


Figure 5.23: Landmark UKR models of a "noisy spiral" dataset. From left to right: $n = 30$, $n = 50$, and $n = 100$. Top row: manifolds after minimizing the reconstruction error w.r.t. the latent variables. The black stars depict the landmark points as generated by K-Harmonic Means. Remaining rows from top to bottom: manifolds after optimizing the landmark points $\hat{\mathbf{Y}}$ with pre-factors $\gamma = 1$, $\gamma = 0.1$, and $\gamma = 0$ of the roughness penalty.

Fish bowl

In a second experiment, we investigated our proposed landmark UKR variant on a problem involving homotopy-based optimization. As a benchmark dataset, we chose the "fish bowl" that we already dealt with in Sec. 3.5. For the experiments here, we generated datasets with 2000 samples each, where we added Gaussian noise with standard deviation $\sigma = 0.5$. In order to fit a landmark UKR model to the data, we first generated n landmark points by use of the K-Harmonic Means algorithm and calculated an initial coarse 2-D projection of the data and landmark points via PCA. Then, we executed 10 homotopy steps where we minimized the penalized reconstruction error (5.41) with a pre-factor $\lambda_k = 0.7^k$ in the k-th homotopy step. Within each step, we carried out up to 1000 RPROP steps. We repeated the experiment 10 times each for 7 different settings of the number of landmark points, where we chose $n = 100, 150, 200, 250, 300, 350$, and 400, and for 2 different density kernels, namely the Gaussian kernel and the finite-support Triweight kernel (5.31). Figure 5.24 depicts one of the generated datasets, as well as corresponding embeddings of a UKR model based on 250 landmark points after the 5th and 10th homotopy step, respectively.

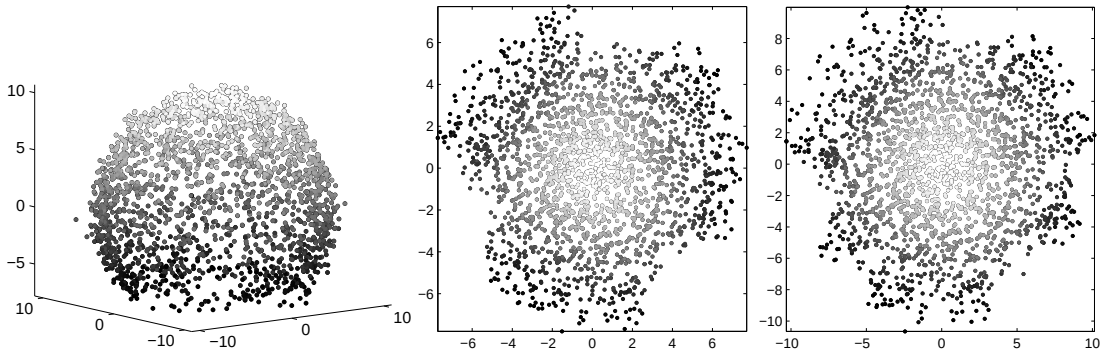


Figure 5.24: An example of a "fish bowl" dataset containing 2000 samples (left plot) and corresponding 2-D embeddings yielded by a UKR model based on the Gaussian kernel and 250 landmark points. The middle plot depicts the model after the 5th homotopy step, while the right plot shows the model after the 10th (and last) homotopy step, thus showing a larger extension of latent space.

We evaluated the resulting 140 models (10 trials $\times$ 7 numbers of landmarks $\times$ 2 kernels) by calculating 1) the mean distance between an original data point $\mathbf{y}_i$ and its reconstruction $\mathbf{f}(\mathbf{x}_i; \hat{\mathbf{X}})$, 2) the mean of the norms $\varrho_i = \|\mathbf{f}(\mathbf{x}_i; \hat{\mathbf{X}})\|$ of the reconstructions, and 3) the standard deviation of that norms. Concerning this, please recall that the underlying "fish bowl" manifold is the clipped surface of an origin-centered sphere with a radius of $\varrho = 10$. Therefore, $|\varrho_i - 10|$ determines the distance between a reconstruction and the true manifold, and the mean of $\{\varrho_i\}$ can serve as a measure of the bias of the model. Similarly, the standard deviation of $\{\varrho_i\}$ can be regarded as a measure of roughness. As a further training step, we adapted the landmark points according to (5.40) with a pre-factor $\gamma = 0.1$, after which we collected the same statistics again.

Table 5.8 summarizes the results for the different models after the 5th and the 10th homotopy step, averaged across 10 trials each. Please note how the landmark adaption improves all aspects of especially the Gaussian-based UKR models: both the recon-

struction error and the roughness is decreased, and the reconstructions lie closer to the surface of the sphere. Most notably, if the homotopy training is stopped after the 5th step, a subsequent landmark adaption produces models with very similar properties to those with the "full" training (10 homotopy steps and landmark adaption).

	landmark points	100	150	200	250	300	350	400
$k = 5$	mean rec. distance	0.97	0.94	0.92	0.91	0.90	0.89	0.89
	mean norm of rec.	9.15	9.16	9.17	9.19	9.19	9.20	9.19
	std. dev. of norms	0.28	0.28	0.28	0.27	0.27	0.27	0.27
$k = 10$	mean rec. distance	0.64	0.61	0.58	0.58	0.57	0.56	0.55
	mean norm of rec.	9.51	9.53	9.55	9.57	9.58	9.59	9.59
	std. dev. of norms	0.16	0.17	0.18	0.18	0.18	0.19	0.19

a) Gaussian kernel, before adaption of landmark points

$k = 5$	mean rec. distance	0.41	0.40	0.38	0.39	0.38	0.39	0.39
	mean norm of rec.	10.02	10.02	10.02	10.03	10.02	10.02	10.01
	std. dev. of norms	0.11	0.12	0.12	0.12	0.12	0.12	0.12
$k = 10$	mean rec. distance	0.41	0.39	0.38	0.38	0.38	0.38	0.37
	mean norm of rec.	10.03	10.02	10.02	10.03	10.02	10.02	10.02
	std. dev. of norms	0.12	0.13	0.14	0.14	0.15	0.15	0.15

b) Gaussian kernel, after adaption of landmark points

$k = 5$	mean rec. distance	0.60	0.55	0.53	0.51	0.49	0.48	0.47
	mean norm of rec.	9.58	9.61	9.64	9.68	9.71	9.75	9.76
	std. dev. of norms	0.17	0.19	0.20	0.24	0.25	0.27	0.27
$k = 10$	mean rec. distance	0.55	0.51	0.47	0.46	0.44	0.43	0.41
	mean norm of rec.	9.66	9.70	9.73	9.77	9.80	9.83	9.87
	std. dev. of norms	0.16	0.18	0.19	0.23	0.24	0.25	0.25

c) Triweight kernel, before adaption of landmark points

$k = 5$	mean rec. distance	0.48	0.49	0.49	0.51	0.51	0.53	0.55
	mean norm of rec.	9.99	9.98	9.97	9.96	9.96	9.95	9.95
	std. dev. of norms	0.14	0.17	0.18	0.24	0.25	0.30	0.31
$k = 10$	mean rec. distance	0.50	0.49	0.50	0.50	0.50	0.50	0.51
	mean norm of rec.	9.99	9.99	9.98	9.97	9.97	9.97	9.98
	std. dev. of norms	0.16	0.19	0.22	0.25	0.28	0.32	0.33

d) Triweight kernel, after adaption of landmark points

Table 5.8: Properties of different landmark UKR models fitted to the "fish bowl" data. The first column indicates the number of homotopy steps the model was trained with. The abbreviations in the second column stand for the three properties 1) mean distance between reconstruction and original data point, 2) mean norm of reconstruction as a measure of the bias, and 3) standard deviation of that norm as a roughness measure. The results are averaged across 10 trials.

As a downside, for the Triweight-based models with a larger number of landmark points the adaption of $\hat{\mathbf{Y}}$ yields reconstructions that are closer to the underlying sphere,

but the roughness and the reconstruction error of the models is slightly increased. Please note, however, that the pre-factors γ and λ were chosen in an ad-hoc manner since we just aimed at a proof of concept, and that it is highly unlikely that we hit the mark for all the different models.

Figure 5.25 depicts the reconstructions $\mathbf{f}(\mathbf{x}_i; \hat{\mathbf{X}})$ of a UKR model with 250 landmark points exemplarily, where again the results after 5 and 10 homotopy steps, as well as before and after the landmark adaption are compared. Please note that the bias towards the inner of the sphere is much larger for the model after the 5th step (upper left plot), but that still the models are comparable after the landmark adaption (right column).

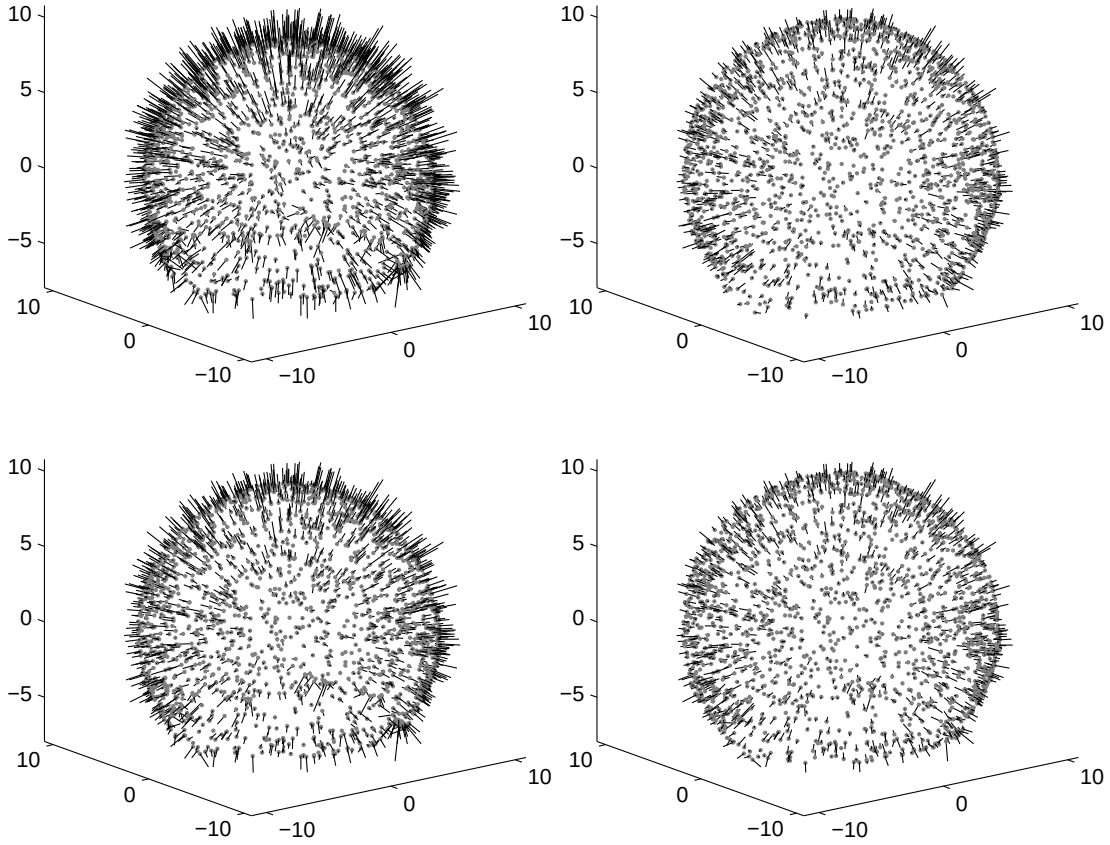


Figure 5.25: Landmark UKR models ($n = 250$) of a "fish bowl" dataset. The models are shown after the 5th (top row) and 10th (bottom row) homotopy step, as well as before (left column) and after (right column) the adaption of landmark points. The reconstructions are depicted by gray dots, with black lines indicating the deviation from the original data points.

Finally, the mean computation time for executing 10 homotopy steps in dependence on the density kernel and the number of landmark points is reported in Table 5.9. Please note that the measured time roughly scales linearly with the number of landmark points, as was expected from the considerations regarding the complexity.

landmark points	100	150	200	250	300	350	400
Gaussian kernel	627	851	1051	1246	1588	1968	2119
Triweight kernel	137	179	227	269	313	356	402

Table 5.9: Computation time in seconds for executing 10 homotopy steps with up to 1000 RPROP steps each. The results are averaged across 10 trials.

5.3.4. Discussion

In this section, we outlined a variant of UKR that first of all aims at reducing the computational complexity. By the concept of reconstructing the data only from a set of previously generated landmark points, training a UKR model can take into account all samples of a possibly large dataset without suffering from a too high computational effort, which normally rises quadratically with the number of training data.

The savings in terms of computation time, however, come at the cost of a less clearly defined learning scheme. Firstly, we have to select a suitable number n of landmark points and an algorithm to generate those points. In a pure vector quantization task, the results of multiple runs with different choices of n can be compared with respect to the quantization error. In our setting, though, the quantization error can only roughly guide the selection of n, since the UKR function (5.22) linearly combines the codebook vectors or landmark points for reconstructing the data. Therefore, assuming suitable latent coordinates $\hat{\mathbf{X}}$ and $\mathbf{X}$ for the landmark and data points, the reconstruction error (5.23) will usually be much smaller than the quantization error, and landmark points that can be convexly combined from other landmark points may be redundant.

Secondly, we have to somehow make sure that the overlap of the density kernels is large enough, since otherwise the UKR mapping (5.22) will show sharp transitions between the landmark points. With this UKR variant, the automatic LOO-CV method cannot be applied anymore, so we have to specify suitable constraints or penalties based on the extension or density in latent space.

Fortunately, our experiments indicate that by an additional adaption of the landmark points $\hat{\mathbf{Y}}$ a similarly shaped manifold can be created from quite different starting conditions, here in the sense of number of landmark points and overlap regularization. In order to achieve this, especially in case of a rather large number of landmark points, we need an additional (third) regularization parameter γ that balances between smoothness and closeness of fit. Since optimizations with respect to $\hat{\mathbf{Y}}$ can be accomplished using linear algebra, different values of γ can quickly be tried out. Then, as a possible way to infer a suitable γ, the resulting UKR manifolds can be compared with regard to the projection error of hold-out data[11].

There are further issues to resolve, for example the choice of sampling points for calculating the roughness metric. Moreover, after an adaption of $\hat{\mathbf{Y}}$, one should determine new projection coordinates $\mathbf{X}$ for the training data $\mathbf{Y}$, and repeat these two steps until convergence. With the same argument as for the Regularized Principal Manifolds (2.6.2) and the PSOM$^+$ (3.5), convergence must occur since a new projection by definition can only decrease the reconstruction error and leave the roughness term un-

[11] Landmark UKR will only be applied in case the training data is plenty, so holding out even larger parts of it should not be a problem.

changed[12], whereas the adaption step minimizes the weighted sum (5.38) of both terms. If we take a further adaption of the kernel centers $\hat{\mathbf{X}}$ into account, however, this does not apply anymore, since the roughness metric depends on these points in a complicated way.

Despite the aforementioned problems, landmark UKR has the undisputed advantage of a much reduced computational cost. Furthermore, the additional adaption of the landmark points $\hat{\mathbf{Y}}$ can yield manifolds which are very smooth and at the same time do not show curvature bias, even if a preceding optimization with respect to the pair $(\mathbf{X}, \hat{\mathbf{X}})$ led to a heavily biased manifold (cf. Fig. 5.23). Therefore, we see landmark UKR as a promising variant and as an interesting direction of future research.

5.4. Unsupervised Local Polynomial Regression

As described in the previous chapter, the UKR algorithm has been derived as an unsupervised counterpart of the Nadaraya-Watson kernel regression estimator. To recall the basic properties, the latter creates a smooth mapping from given sets of input and output data, denoted by $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_N] \in \mathbb{R}^{q \times N}$ and $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2 \dots \mathbf{y}_N] \in \mathbb{R}^{d \times N}$, respectively. In particular, the Nadaraya-Watson estimator forms a locally weighted convex combination

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) = \sum_i \mathbf{y}_i \frac{K_h(\mathbf{x} - \mathbf{x}_i)}{\sum_j K_h(\mathbf{x} - \mathbf{x}_j)} \tag{5.42}$$

of the output data. In the UKR setting, $\mathbf{X}$ are treated as parameters, and the optimal set of these is determined by minimizing the reconstruction error (4.26) or rather a penalized (4.31,4.33) or cross-validated version (4.34) of it.

In section 4.1 we showed how to derive the Nadaraya-Watson estimator from a kernel density estimate of the conditional expectation, but one can also view the estimator from a different perspective. As will be demonstrated later, the Nadaraya-Watson estimator is the simplest example from the class of local polynomial regression estimators, which aim at fitting a polynomial $\mathbf{f}_\mathbf{x}(\cdot)$ at each point $\mathbf{x}$ to the data. Early work on local polynomial regression has been carried out by C. J. Stone (1977) and Cleveland (1979). A more recent review of the technique including a historical overview has been given by Cleveland and Loader (1996).

This section of the thesis is aimed at the investigation how also more complex members of that class can be transformed into unsupervised manifold learning methods, resulting in the new family of Unsupervised Local Polynomial Regression (ULPR) algorithms. As with Landmark UKR, the ideas and results presented here have only recently been developed and not been published, yet.

In the remainder of this section, we first give a brief introduction to classical (supervised) local polynomial regression. Then, we will derive unsupervised variants in direct analogy to the way we transformed the Nadaraya-Watson estimator into UKR. As with previous variants of UKR, we will demonstrate the properties of the new algorithms in a series of experiments, and we will conclude this section with a discussion of the results of the experiments.

[12] To this effect, we must not change the sampling points, which would change the roughness metric.

5.4.1. Local polynomial regression

A simple global approach to learning a relationship between input and output data is to optimize a set of parameters $\boldsymbol{\Theta}$ of a function $\mathbf{f}(\cdot;\boldsymbol{\Theta})$, so that the empirical error

$$E(\boldsymbol{\Theta}) = \sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i;\boldsymbol{\Theta})\|^2, \tag{5.43}$$

or a regularized variant of it, becomes as small as possible. Here Θ might be the coefficients of a high-dimensional polynomial or a Fourier series, or the weights of a multi-layer perceptron. Either way, one fits *one global*, possibly very complex function to the complete dataset.

In contrast to this, *local* regression aims at fitting multiple simple functions to the data, that all are suitable only in a small region. For this, kernel functions $K_h(\cdot)$ are used to weight the influence of the data points to the regression error of a local estimator $\mathbf{f}_\mathbf{x}(\cdot)$ centered at $\mathbf{x}$:

$$E(\mathbf{f}_\mathbf{x}) = \sum_{i=1}^{N} K_h(\mathbf{x} - \mathbf{x}_i)\|\mathbf{y}_i - \mathbf{f}_\mathbf{x}(\mathbf{x}_i)\|^2. \tag{5.44}$$

Here, a bandwidth parameter h (cf. Sec. 4.1) determines the size of the region where the local approximation should hold. In the limit of an infinite bandwidth, the local regression estimate has to take all data samples into account, so it effectively becomes a global regression model. For smaller bandwidths, and especially for a kernel $K_h(\cdot)$ with finite support, the local model has to fit only a few data points well, which thus can be done more closely.

Since any smooth function can locally be approximated by its Taylor expansion, polynomials make up a suitable function class for the local estimators $\mathbf{f}_\mathbf{x}(\cdot)$. In this work, we restrict ourselves to the three simplest cases, that is, we consider local constant regression (polynomials of degree zero)

$$\mathbf{f}_\mathbf{x}(\mathbf{x}_i) = \mathbf{m}_\mathbf{x}, \tag{5.45}$$

where the function value $\mathbf{m}_\mathbf{x} \in \mathbb{R}^d$ is independent of the argument $\mathbf{x}_i$, local linear regression (degree one)

$$\mathbf{f}_\mathbf{x}(\mathbf{x}_i) = \mathbf{m}_\mathbf{x} + \mathbf{G}_\mathbf{x}(\mathbf{x}_i - \mathbf{x}) = \mathbf{m}_\mathbf{x} + \sum_{\mu=1}^{q} \mathbf{g}_\mathbf{x}^\mu (\mathbf{x}_i - \mathbf{x})_\mu, \tag{5.46}$$

where $\mathbf{G}_\mathbf{x} = (\mathbf{g}_\mathbf{x}{}^1, \mathbf{g}_\mathbf{x}{}^2 \ldots \mathbf{g}_\mathbf{x}{}^q) \in \mathbb{R}^{d\times q}$, and local quadratic regression (degree two)

$$\mathbf{f}_\mathbf{x}(\mathbf{x}_i) = \mathbf{m}_\mathbf{x} + \mathbf{G}_\mathbf{x}(\mathbf{x}_i - \mathbf{x}) + \sum_{\mu=1}^{q} \sum_{\substack{\nu=1 \\ \nu\le\mu}}^{q} \mathbf{h}_\mathbf{x}^{\mu\nu} (\mathbf{x}_i - \mathbf{x})_\mu (\mathbf{x}_i - \mathbf{x})_\nu. \tag{5.47}$$

At first, we take a closer look at the local constant case, and we calculate the optimal function value $\mathbf{m}_\mathbf{x}$ for the local estimator at $\mathbf{x}$. To this end, we insert (5.45) into (5.44) and minimize the resulting expression

$$E(\mathbf{m}_\mathbf{x}) = \sum_{i=1}^{N} K_h(\mathbf{x} - \mathbf{x}_i)\|\mathbf{y}_i - \mathbf{m}_\mathbf{x}\|^2 \tag{5.48}$$

with respect to $\mathbf{m_x}$, yielding

$$\mathbf{0} \stackrel{!}{=} \nabla E(\mathbf{m_x}) = -2\sum_{i=1}^{N} K_h(\mathbf{x}-\mathbf{x}_i)(\mathbf{y}_i - \mathbf{m_x}) \tag{5.49}$$

$$\Rightarrow \sum_{i=1}^{N} K(\mathbf{x}-\mathbf{x}_i)\mathbf{m_x} = \sum_{i=1}^{N} K(\mathbf{x}-\mathbf{x}_i)\mathbf{y}_i \tag{5.50}$$

$$\Rightarrow \mathbf{m_x} = \sum_{i=1}^{N} \mathbf{y}_i \frac{K(\mathbf{x}-\mathbf{x}_i)}{\sum_{j=1}^{N} K(\mathbf{x}-\mathbf{x}_j)}. \tag{5.51}$$

As a result, we see that the optimal *local constant fit* equals the Nadaraya-Watson estimator. To obtain the optimal *local linear fit*, we first define the "design matrix"

$$\bar{\mathbf{X}} = (\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2 \dots \bar{\mathbf{x}}_N) = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \mathbf{x}_1 - \mathbf{x} & \mathbf{x}_2 - \mathbf{x} & \dots & \mathbf{x}_N - \mathbf{x} \end{pmatrix} \in \mathbb{R}^{(q+1)\times N} \tag{5.52}$$

as well as a diagonal matrix $\mathbf{W} \in \mathbb{R}^N$ with entries

$$w_{ii} = K_h(\mathbf{x}_i - \mathbf{x}). \tag{5.53}$$

If we further define $\mathbf{M_x} = (\mathbf{m_x}, \mathbf{G_x}) = (\mathbf{m_x}, \mathbf{g}_\mathbf{x}^1 \dots \mathbf{g}_\mathbf{x}^q)$, we can reformulate (5.44) as

$$E(\mathbf{M_x}) = \sum_{i=1}^{N} w_{ii} \|\mathbf{y}_i - \mathbf{M_x}\bar{\mathbf{x}}_i\|^2. \tag{5.54}$$

As with the local constant case, the minimum of (5.54) is given by the zero of its derivative with respect to $\mathbf{M_x}$, which yields

$$\mathbf{0} \stackrel{!}{=} \frac{\partial E(\mathbf{M_x})}{\partial \mathbf{M_x}} = -2\sum_{i=1}^{N} w_{ii}(\mathbf{y}_i - \mathbf{M_x}\bar{\mathbf{x}}_i)\bar{\mathbf{x}}_i^T = -2(\mathbf{Y} - \mathbf{M_x}\bar{\mathbf{X}})\mathbf{W}\bar{\mathbf{X}}^T \tag{5.55}$$

$$\Rightarrow \mathbf{M_x} = \mathbf{Y}\mathbf{W}\bar{\mathbf{X}}^T \left(\bar{\mathbf{X}}\mathbf{W}\bar{\mathbf{X}}^T\right)^{-1}. \tag{5.56}$$

In principle, the same calculation also works for local quadratic regression, but since the number of variables rises quadratically with the dimensionality of the input space, the notation gets a bit messy. For the case of one-dimensional inputs x_i, for example, we can define

$$\bar{\mathbf{X}} = (\bar{\mathbf{x}}_1, \bar{\mathbf{x}}_2 \dots \bar{\mathbf{x}}_N) = \begin{pmatrix} 1 & 1 & \dots & 1 \\ x_1 - x & x_2 - x & \dots & x_N - x \\ (x_1 - x)^2 & (x_2 - x)^2 & \dots & (x_N - x)^2 \end{pmatrix} \in \mathbb{R}^{3\times N} \tag{5.57}$$

and $\mathbf{M_x} = (\mathbf{m_x}, \mathbf{g_x}, \mathbf{h_x})$, by which (5.56) holds again.

Please note that if we fit a local model at each point $\mathbf{x}$ of interest (e.g., for prediction), we actually only need to calculate $\mathbf{m_x}$ even if we wish to use linear or quadratic regression, since for example in the linear case we have

$$\mathbf{f_x}(\mathbf{x}) = \mathbf{m_x} + \mathbf{G_x}(\mathbf{x} - \mathbf{x}) = \mathbf{m_x}. \tag{5.58}$$

The smoothness and complexity of the complete (global) regression function $\mathbf{f}(\mathbf{x}) = \mathbf{f}_{\mathbf{x}}(\mathbf{x})$ depends on 1) the degree of the local polynomials and 2) the bandwidth h used to fit the local estimators. A larger bandwidth gives larger weight to a wider span of data samples, so estimators at nearby points $\mathbf{x}$ and $\mathbf{x}'$ differ less, which yields a smoother global estimator. A higher degree of the polynomials implies a higher variability of the local estimators and therefore also of the global function.

As an example, Fig. 5.26 shows the results of local constant, linear and quadratic regression for two different bandwidths and a Gaussian kernel, where the task is to learn a simple function from 100 noisy samples of a sinus graph ($q = d = 1$). Note the much better fit of the linear and quadratic estimators for the larger bandwidth (left column) and especially the better behavior at the borders. Even for the smaller bandwidth, the local constant (Nadaraya-Watson) fit shows an undesirable bias at the ends of the curve. Please note also that at the peaks of the sinus function, where the curvature is high, the local quadratic estimator produces a more accurate fit than the linear estimator. These properties strongly motivate to generalize the UKR idea to an underlying local polynomial regression estimate.

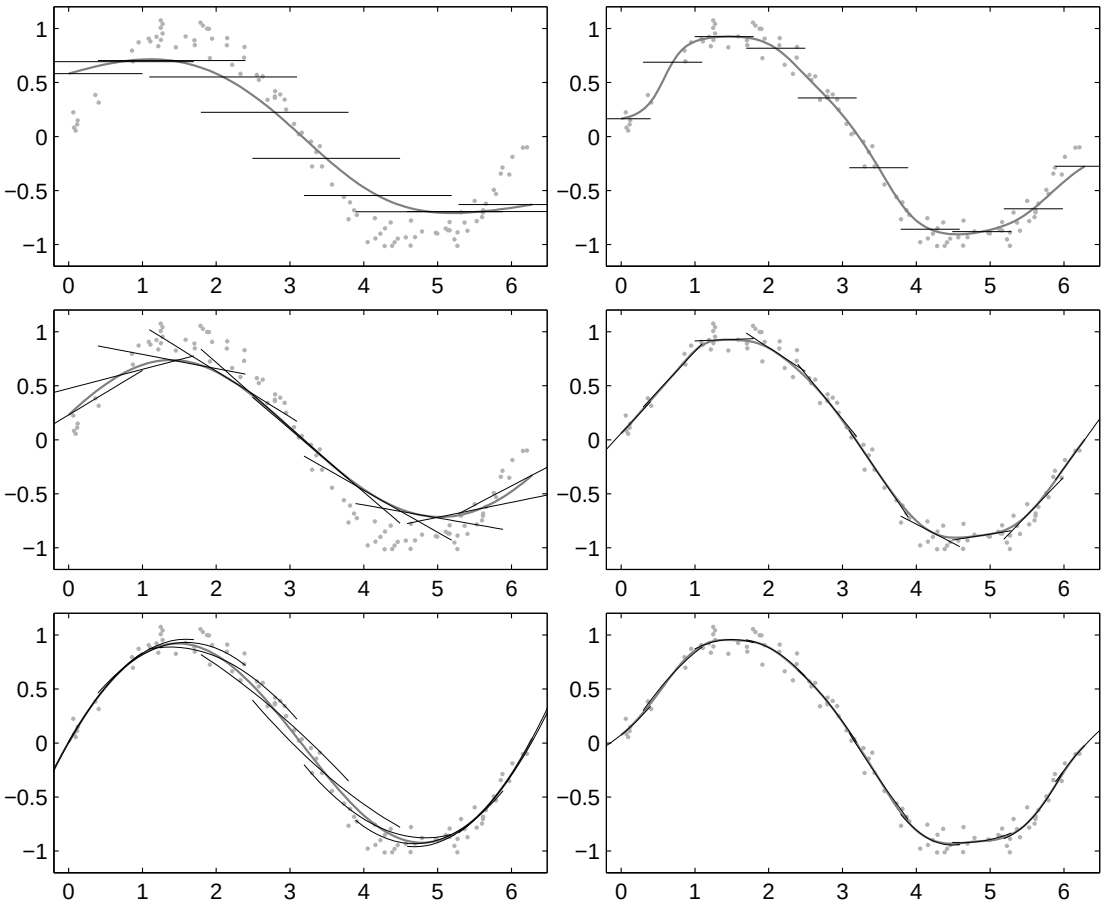


Figure 5.26: Using Local Polynomial Regression to learn a function from noisy data. From top to bottom: local constant fit (Nadaraya-Watson), local linear fit, local quadratic fit. Left: bandwidth $h = 1$. Right: bandwidth $h = 0.3$. The black curves depict 10 local estimators each.

Computational effort

Before formulating an unsupervised version of local polynomial regression, we turn our attention to the computational effort that is involved in one evaluation of the regression function

$$\mathbf{f}(\mathbf{x}) = \mathbf{m}_\mathbf{x} = \mathbf{M}_\mathbf{x}\mathbf{e}_0 = \mathbf{Y}\mathbf{W}\bar{\mathbf{X}}^T \left(\bar{\mathbf{X}}\mathbf{W}\bar{\mathbf{X}}^T\right)^{-1} \mathbf{e}_0 = \mathbf{Y}\mathbf{W}\bar{\mathbf{X}}^T\mathbf{C}^{-1}\mathbf{e}_0, \tag{5.59}$$

where $\mathbf{C} = \bar{\mathbf{X}}\mathbf{W}\bar{\mathbf{X}}^T$ and $\mathbf{e}_0 = (1, 0, \ldots, 0)^T$ is the first standard basis vector. The length n of the vector $\mathbf{e}_0$ depends on the dimensionality q of the input data and on the choice of the algorithm. The same applies to the $n \times n$ matrix $\mathbf{C}$ and the design matrix $\bar{\mathbf{X}} \in \mathbb{R}^{n \times N}$. This n is given by

$$\begin{array}{rl} n = 1 & \text{local constant regression (Nadaraya-Watson),} \\ n = q + 1 & \text{local linear regression, and} \\ n = \frac{1}{2}(q+1)(q+2) & \text{local quadratic regression.} \end{array} \tag{5.60}$$

For one evaluation of (5.59), we need

- $O(N)$ operations to calculate the diagonal elements of $\mathbf{W}$ from the distances $d_i = \|\mathbf{x} - \mathbf{x}_i\|^2$ and $O(qN)$ operations to calculate these distances,
- $O(n^2N)$ operations to calculate $\mathbf{C} = \bar{\mathbf{X}}\mathbf{W}\bar{\mathbf{X}}^T$ (recall that $\mathbf{W}$ is diagonal),
- $O(n^3)$ operations to invert $\mathbf{C}$, or less if we use the fact to we only need the first column,
- $O(nN)$ operations to calculate $\mathbf{a} = \bar{\mathbf{X}}^T(\mathbf{C}^{-1}\mathbf{e}_0)$,
- $O(N)$ operations to calculate $\mathbf{b} = \mathbf{W}\mathbf{a} = \mathbf{W}\bar{\mathbf{X}}^T\mathbf{C}^{-1}\mathbf{e}_0$ and finally
- $O(dN)$ operations to calculate $\mathbf{m}_\mathbf{x} = \mathbf{Y}\mathbf{b}$.

If $d \gg q$ (the case we are interested in for dimensionality reduction), the most expensive operation is the $O(dN)$ multiplication $\mathbf{Y}\mathbf{b}$, which is independent of the degree of the local polynomials. If d is not much bigger than q, however, the $O(n^2N)$ operations can get quite a burden for the case of local quadratic regression, where $O(n^2N)$ means $O(q^4N)$.

Please note that also the general local polynomial regression function can be written in the form "data matrix $\times$ vector of basis functions", where now the basis functions are given by

$$\mathbf{b}(\mathbf{x}; \mathbf{X}) = \mathbf{W}\bar{\mathbf{X}}^T\mathbf{C}^{-1}\mathbf{e}_0. \tag{5.61}$$

5.4.2. Derivation of ULPR

The transformation of local linear and quadratic regression into unsupervised manifold learning algorithms works in the same way as for the Nadaraya-Watson estimator. We

just treat the input data $\mathbf{X}$ as latent variables, which we optimize by minimizing the reconstruction error

$$R(\mathbf{X}) = \frac{1}{N} \sum_i \|\mathbf{y}_i - \mathbf{f}(\mathbf{x}_i; \mathbf{X})\|^2 = \frac{1}{N} \sum_i \|\mathbf{y}_i - \mathbf{Y}\mathbf{b}(\mathbf{x}_i; \mathbf{X})\|^2. \tag{5.62}$$

Again, the bandwidth parameter can be canceled out since the scaling of $\mathbf{X}$ is free. We can use the already known regularization approaches from standard UKR, most notably LOO-CV and also LKO-CV (Sec. 5.2) without additional computational cost. Please note, however, that e.g., for LOO-CV the relevant diagonal elements $K_{ii} = K(\mathbf{x}_i - \mathbf{x}_i)$ have to be set to zero already for calculating the matrices $\mathbf{W}$ and $\mathbf{C}$.

Because of the matrix inversion involved in the local linear and quadratic regression functions, the analytic calculation of the gradient of $R(\mathbf{X})$ is more complicated than for UKR. Please see appendix C for a detailed derivation, and also for the particular form of the gradients for the different degrees of the local polynomials. For simplicity, we analyzed the "full" local quadratic estimate only for the case $q = 2$, where the j-th column of the design matrix involved in calculating $\mathbf{b}(\mathbf{x}_i; \mathbf{X})$ is given by

$$\bar{\mathbf{x}}_j^i = \begin{pmatrix} 1 \\ x_{1j} - x_{1i} \\ x_{2j} - x_{2i} \\ (x_{1j} - x_{1i})^2 \\ (x_{2j} - x_{2i})^2 \\ (x_{1j} - x_{1i})(x_{2j} - x_{2i}) \end{pmatrix} \in \mathbb{R}^6. \tag{5.63}$$

For the local quadratic case and a general latent dimensionality q, we left out cross-terms in the design matrix, that is, terms like $(x_{1j} - x_{1i})(x_{2j} - x_{2i})$. The same simplification has been applied by Yang and Tschernig (1999), who investigated using local quadratic regression in the problem of bandwidth selection for kernel density estimation. Please note that the number n of rows of the design matrix equals $n = 2q + 1$ if no cross-terms are included, and that there are no cross-terms in the case $q = 1$.

Please also note that it is possible to modify (5.62) by the inclusion of a general loss function in the way of Sec. 5.1. The local estimators are explicitly based on the squared loss, but still a different global loss function will influence the way the ULPR manifold is fitted, as we have already seen in the local constant case (UKR) in Sec. 5.1.4.

5.4.3. Experiments

Half circle

In a first experiment, we fitted ULPR models of varying polynomial degree to the synthetic "half circle" dataset already known from section 5.1.4. Here, we only took into account the 4×100 datasets containing Gaussian noise, examples of which can be found in the bottom row of Fig. 5.2, and we only utilized the standard squared loss.

Just like in Sec. 5.1.4, we initialized the models with a PCA solution (scaled down to a variance of 1) and then carried out up to 2000 RPROP steps minimizing the CV-error. In addition to a pure LOO-CV regularization, here we also added the extension penalty term scaled by pre-factors $\lambda = 0.01$, $\lambda = 0.1$ and $\lambda = 1$. In total, we thus fitted 4800

models, corresponding to 100 trials with 4 different noise levels, 3 polynomial degrees (constant, linear, and quadratic), and 4 different regularization strategies (pure LOO-CV, 3× LOO-CV with additional extension penalty). As the basis for all models, we used the Gaussian density kernel.

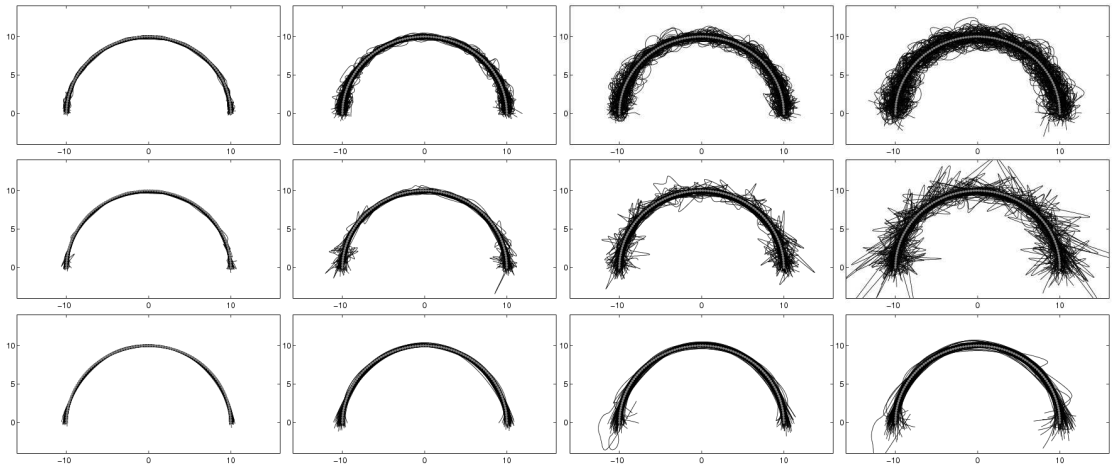


Figure 5.27: ULPR models of noisy half circle datasets. From top to bottom: locally constant (UKR), linear, and quadratic. From left to right: $\sigma = 0.25$, $\sigma = 0.5$, $\sigma = 0.75$, $\sigma = 1$. All models are based on the Gaussian kernel and are regularized solely by LOO-CV. The plots each depict 100 models drawn atop each other. The dotted gray line indicates the underlying half circle.

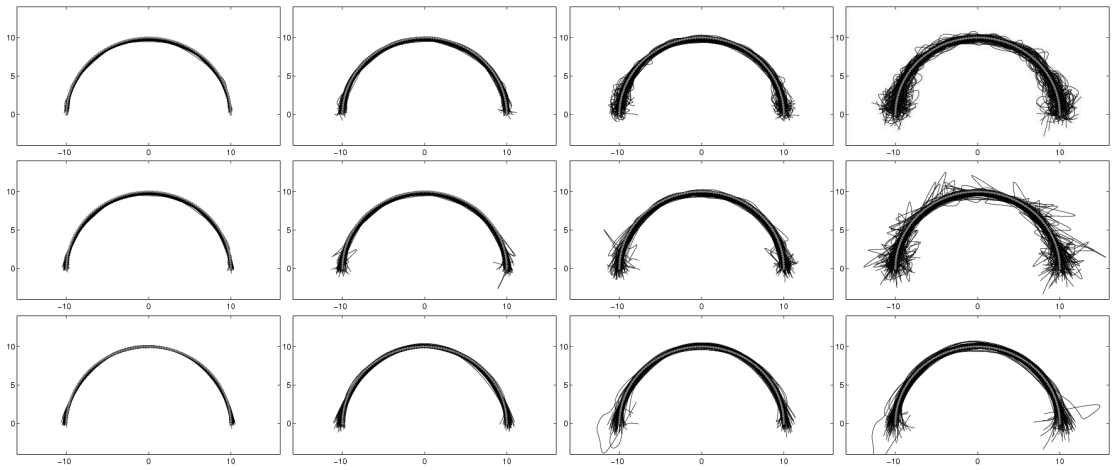


Figure 5.28: ULPR models of noisy half circle datasets, regularized by LOO-CV with an additional extension penalty ($\lambda = 0.01$). From top to bottom: locally constant (UKR), linear, and quadratic. From left to right: $\sigma = 0.25$, $\sigma = 0.5$, $\sigma = 0.75$, $\sigma = 1$.

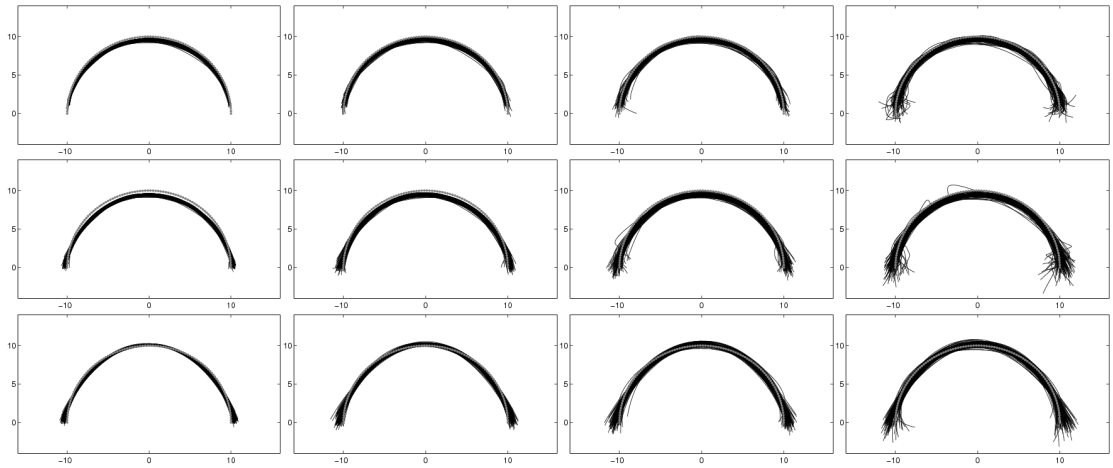


Figure 5.29: ULPR models of noisy half circle datasets, regularized by LOO-CV with an additional extension penalty ($\lambda = 0.1$). From top to bottom: locally constant (UKR), linear, and quadratic. From left to right: $\sigma = 0.25$, $\sigma = 0.5$, $\sigma = 0.75$, $\sigma = 1$.

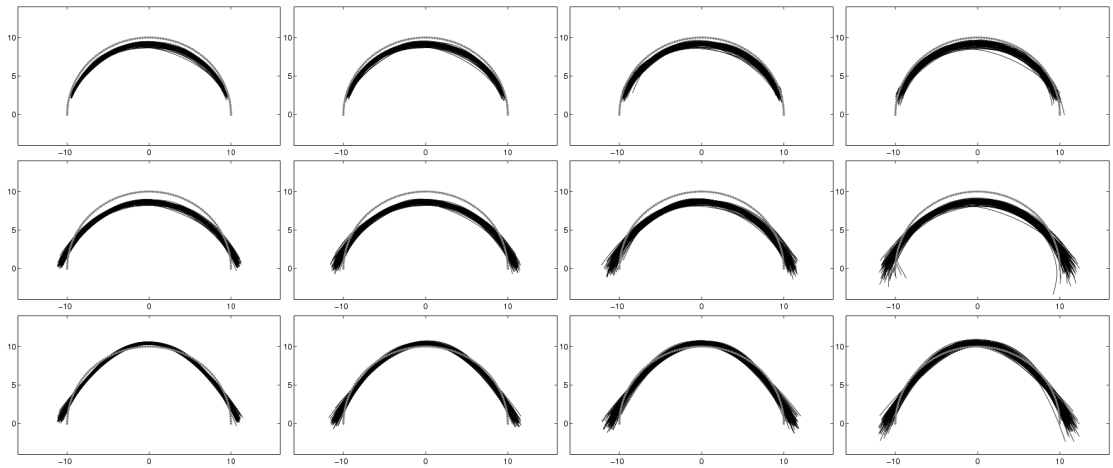


Figure 5.30: ULPR models of noisy half circle datasets, regularized by LOO-CV with an additional extension penalty ($\lambda = 1$). From top to bottom: locally constant (UKR), linear, and quadratic. From left to right: $\sigma = 0.25$, $\sigma = 0.5$, $\sigma = 0.75$, $\sigma = 1$.

The results of this experiment are illustrated by Fig. 5.27 – 5.30, from which we can see several things:

- Both the local constant (UKR) and local linear models show an increasing curvature bias if they are regularized more strongly (Fig. 5.29 and Fig. 5.30). The local quadratic models (depicted in the bottom row of all figures) do not share this drawback – the corresponding (black) curves are centered at the "true" underlying half circle (gray dots). Only the ends of the curves are turned slightly outwards, which is best seen in Fig. 5.30 ($\lambda = 1$).

- The local linear models seem to be particularly bad. They show as much (or even more) bias as the local constant models, and in addition they yield very wriggly

curves if the noise level is high and only a mild regularization is applied (Fig. 5.27 and 5.28).

- Even for high noise levels and a pure LOO-CV regularization, the local quadratic models produce smooth curves, as is depicted by the bottom right plot in Fig. 5.27. On one hand, this is understandable because the built-in quadratic fit matches the half circle problem well. On the other hand, a local quadratic model has much more variability than a local constant model, and should therefore lead to *less* smooth curves which tend to interpolate the data. As a possible explanation, the local quadratic models may all have gotten stuck in a benign local minimum early in the optimization process, where the latent variables were still close and thus the manifold still smooth. If in such an early stadium the data can already be fitted well, the corresponding minima are deeper and therefore harder to avoid than in the local constant (standard UKR) case. While this viewpoint is a bit unsatisfying, please keep in mind that the aforementioned plot depicts the results of 100 trials, nearly all of them leading to a reasonable curve.

Noisy spiral

In a second experiment, we recycled the 100 "noisy spiral" datasets with uniform noise and also the corresponding Isomap initializations from section 5.1.4. We then fitted 36 different ULPR models to each of the 100 datasets, where we compared the three different density kernels Gaussian, Quartic, and Triweight, the three different polynomial degrees constant, linear, and quadratic, two loss functions (Huber and L_2), and two regularization approaches (LOO-CV and LKO-CV with $K = 5$). Unlike our experiments in Sec. 5.2, we did *not* include a density variance penalty term to cope with wriggly curve ends.

To run this experiment, we again utilized our own UKR toolbox for MATLAB, which allows the user to adjust the desired model properties by simply specifying certain elements of a data structure used to encapsulate a UKR/ULPR model. Then, for evaluating gradients and the regression function, for example, one can call the same toolbox functions regardless of the particular type of model.

As already described in previous experiments, for fitting each of the 3600 models we scale-optimized and compared the Isomap solutions with respect to the LOO-CV or LKO-CV error, selected the best candidate, and carried out up to 1000 RPROP steps for fine-tuning the models. We compared the final models with regard to the mean distance between the UKR/ULPR reconstructions $\mathbf{f}(\mathbf{x}_i; \mathbf{X})$ and the underlying spiral. The results, averaged across all 100 datasets, are summarized in Table 5.10.

Please note that regardless of the density kernel, the local quadratic models with the L_2 loss function and LKO-CV regularization ($K = 5$) provided the best results. The models based on locally linear estimates are worse than the standard UKR (local constant) models in most cases, which again can be explained by the fact that a local linear approximation does not improve the fit in the presence of curvature. A visualization of one of the 100 datasets and all corresponding 36 models is given by Fig. 5.31, 5.32, and 5.33.

	Huber's loss LOO-CV	L_2 loss LOO-CV	Huber's loss LKO-CV (5)	L_2 loss LKO-CV (5)
Local constant fit, Gaussian kernel	0.0299 0.0060 -0.0007	0.0291 0.0029 -0.0015	0.0204 0.0135 -0.0119	0.0196 0.0115 -0.0101
Local linear fit, Gaussian kernel	0.0261 0.0054 -0.0026	0.0304 0.0067 -0.0021	0.0217 0.0130 -0.0132	0.0213 0.0093 -0.0112
Local quadratic fit, Gaussian kernel	0.0217 0.0072 0.0012	0.0206 0.0072 0.0010	0.0161 0.0116 -0.0036	**0.0151** 0.0094 **-0.0015**
Local constant fit, Quartic kernel	0.0293 0.0067 -0.0008	0.0281 0.0043 -0.0018	0.0196 0.0115 -0.0112	0.0181 0.0065 -0.0097
Local linear fit, Quartic kernel	0.0242 0.0057 -0.0018	0.0279 0.0062 -0.0017	0.0217 0.0103 -0.0144	0.0211 0.0121 -0.0123
Local quadratic fit, Quartic kernel	0.0213 0.0065 0.0015	0.0201 0.0083 0.0016	0.0167 0.0141 -0.0040	**0.0151** 0.0097 **-0.0013**
Local constant fit, Triweight kernel	0.0307 0.0050 -0.0003	0.0302 0.0048 -0.0013	0.0198 0.0110 -0.0107	0.0188 0.0072 -0.0088
Local linear fit, Triweight kernel	0.0239 0.0064 -0.0026	0.0275 0.0068 -0.0024	0.0219 0.0115 -0.0144	0.0210 0.0110 -0.0124
Local quadratic fit, Triweight kernel	0.0214 0.0071 0.0013	0.0199 0.0070 0.0015	0.0167 0.0142 -0.0041	**0.0152** 0.0100 **-0.0015**

Table 5.10: ULPR models of "noisy spiral" datasets, compared by the mean distance between data reconstructions and the underlying spiral model. The results are averaged across 100 datasets, where the first two numbers in each cell denote the mean and the standard deviation. The 3rd entry stands for the mean *signed* distance, or *bias* of the reconstructions, with negative values denoting a bias towards the inner of the spiral. Within each block corresponding to a particular density kernel, bold numbers indicate the most favorable combination of reconstruction error and bias.

In this experiment, where Isomap provides very good initializations in most cases, one can hardly argue that the smooth curves of the local quadratic models are just the product of getting stuck in an early local minimum. We can rather conclude that indeed the better fitting properties of local quadratic regression are responsible for the more truthful models of the complete dataset. In particular, please note how the local quadratic models show practically no bias, as printed in the 3rd row of each cell in Table 5.10, even if LKO-CV with $K = 5$ is utilized for generating smoother manifolds. The bias of both the local constant and the local linear models is substantially higher in that case.

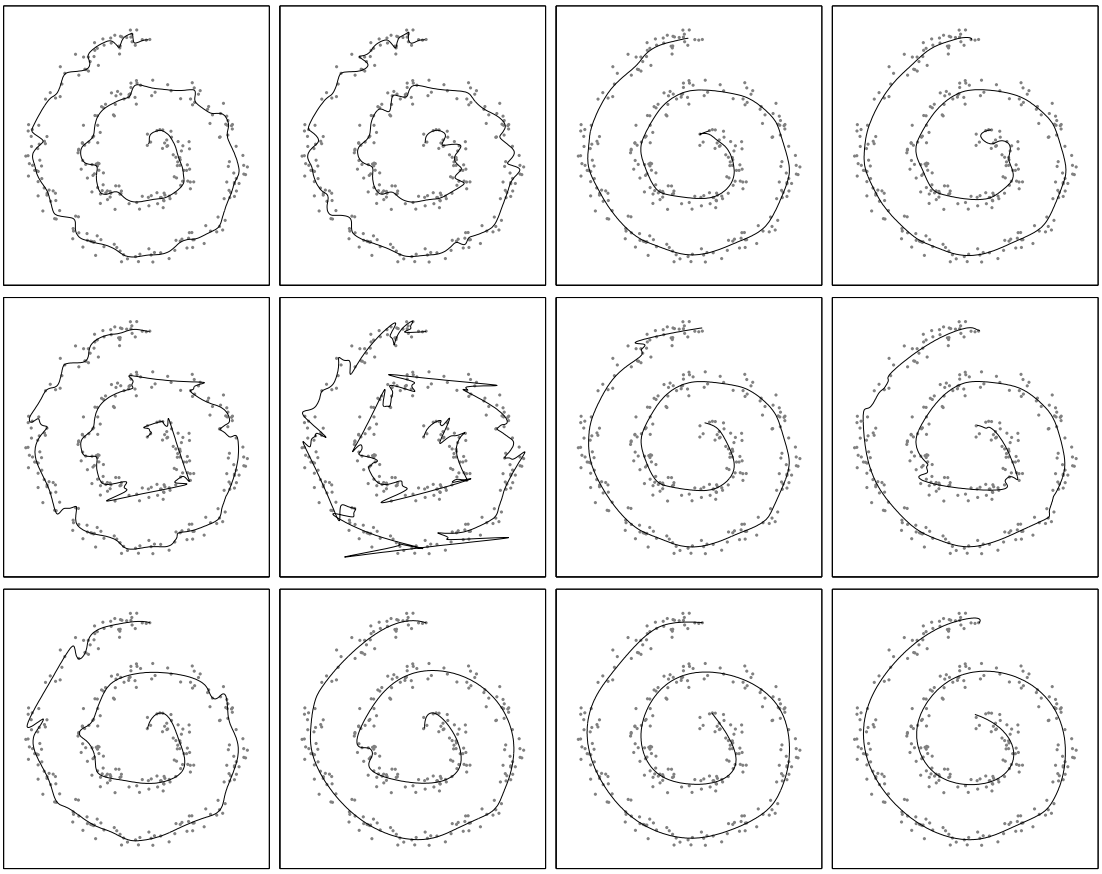


Figure 5.31: Gaussian kernel ULPR models of a "noisy spiral" dataset, with the plots corresponding to the layout of Table 5.10, that is, from top to bottom: local constant, linear, and quadratic model. From left to right: LOO-CV with Huber's loss, LOO-CV with L_2 loss, LKO-CV (K=5) with Huber's loss, LKO-CV (K=5) with L_2 loss.

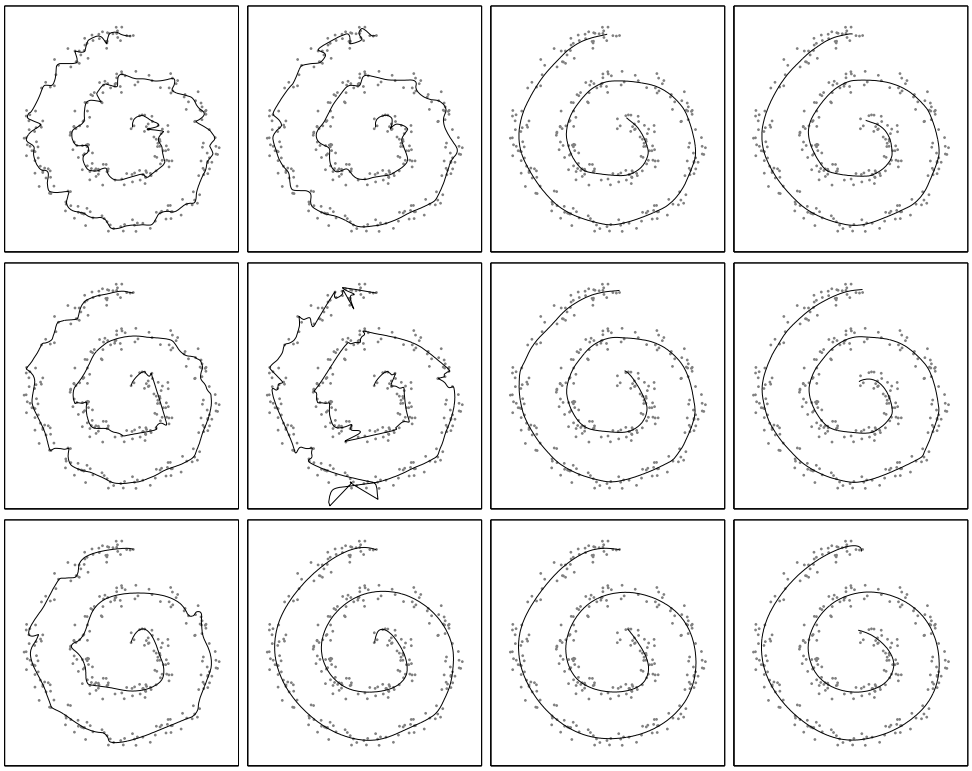


Figure 5.32: Quartic kernel ULPR models of the same datasets as in Fig. 5.31.

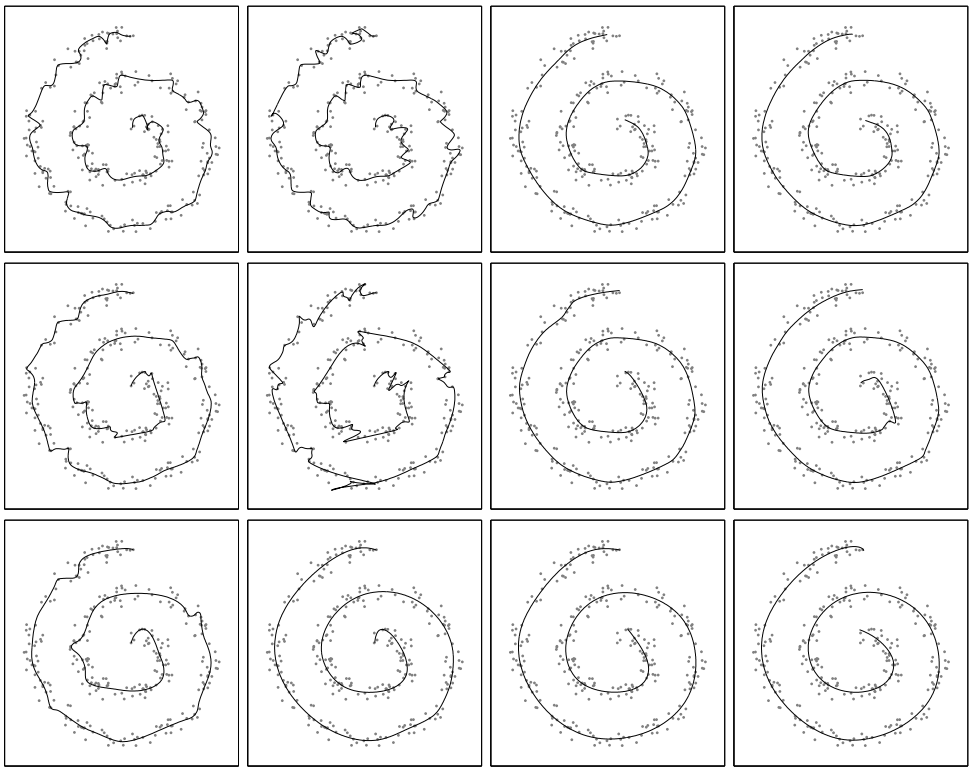


Figure 5.33: Triweight kernel ULPR models of the same datasets as in Fig. 5.31.

Swiss Roll

In a third and last experiment, we fitted local linear and quadratic ULPR models based on the Quartic kernel to samples of the "Swiss Roll". We utilized the same 2×100 datasets containing Gaussian and Laplacian noise that we already handled in Sec. 5.1.4, and we also "recycled" the initial solutions provided by Isomap. We fitted 6 models to each of the 200 datasets, combining 1) the local linear estimators, 2) the local quadratic estimators without the cross-term, and 3) the full quadratic estimators with a) Huber's loss and b) the standard L_2 loss. As described before, we first optimized the scales of the Isomap solutions with respect to the LOO-CV error (measured with the respective loss function), and then further fine-tuned the model by minimizing that error as a function of the latent variables in up to 1000 RPROP steps.

In accordance to the "Swiss Roll" experiment in Sec. 5.1.4, we evaluated the final models by measuring the mean (absolute) distance between the reconstructions $\mathbf{f}(\mathbf{x}_i; \mathbf{X})$ and the underlying smooth manifold. Additionally, we took into account whether the reconstructions lied more towards the inner or the outer of the underlying true manifold, for which we evaluated *signed* distances similar to the "noisy spiral" experiment.

The results are given in Table 5.11, where we also included the relevant measurements for the local constant (standard UKR) models that we already fitted in the corresponding experiment of Sec. 5.1.4.

type	degree	L_2 loss	Huber's loss
absolute distance Gaussian noise	local constant (UKR)	0.281 ± 0.020	0.236 ± 0.020
	local linear	0.238 ± 0.029	0.233 ± 0.049
	quad. w/o cross-terms	0.231 ± 0.027	0.190 ± 0.024
	local quadratic	0.214 ± 0.028	0.176 ± 0.022
signed distance Gaussian noise	local constant (UKR)	-0.026 ± 0.018	-0.027 ± 0.019
	local linear	-0.089 ± 0.036	-0.146 ± 0.066
	quad. w/o cross-terms	0.005 ± 0.016	0.005 ± 0.017
	local quadratic	0.002 ± 0.017	0.002 ± 0.017
absolute distance Laplacian noise	local constant (UKR)	0.266 ± 0.022	0.225 ± 0.018
	local linear	0.231 ± 0.034	0.220 ± 0.054
	quad. w/o cross-terms	0.216 ± 0.029	0.179 ± 0.025
	local quadratic	0.200 ± 0.028	0.168 ± 0.022
signed distance Laplacian noise	local constant (UKR)	-0.020 ± 0.020	-0.021 ± 0.019
	local linear	-0.083 ± 0.039	-0.130 ± 0.074
	quad. w/o cross-terms	0.004 ± 0.021	0.003 ± 0.020
	local quadratic	0.001 ± 0.022	0.002 ± 0.022

Table 5.11: Mean distance between the underlying smooth "Swiss Roll" and the UKR or ULPR reconstructions of the training data. Results are averaged across 100 datasets, and printed in the form *mean* ± *standard deviation*. All models were regularized by LOO-CV. Please note again that the signed distance serves as a measure of the bias, with negative values indicating a bias towards the inner of the "Swiss Roll".

Please note that for this problem, where curvature is present only along one of the two intrinsic dimensions (cf. Fig. 5.10 on page 113), already the local linear models yield

a reduced absolute distance to the true manifold on average. If Huber's loss function is utilized, however, this reduction is smaller than the (increased) standard deviation and therefore not significant. The local quadratic models, both with and without the cross-term, yield a significantly reduced absolute distance to the true manifold for both loss functions. In direct comparison, the quadratic model with the cross-term yields slightly (albeit not significantly) better results than the model without the cross-term.

If we look at the *signed* distances as a measure of the bias, we see that the local linear models exhibit a large bias towards the inner of the "Swiss Roll", while both kinds of quadratic models practically show no bias on average.

As a follow-up, we repeated the experiment under the replacement of the LOO-CV regularization by LKO-CV (Sec. 5.2) with an ad-hoc choice of $K = 3$. We did not include a density variance penalty. In this case, contrary to our previous observations, utilizing LKO-CV actually worsens the results (see Table 5.12) as compared to the models regularized by LOO-CV (Table 5.11). Still, within the set of models regularized by LKO-CV, utilizing local quadratic regression yields a smaller distance to the underlying smooth manifold than local constant (standard UKR) models. Interestingly, the advantage of the local quadratic models is bigger if Huber's loss function is utilized, where both the absolute and the signed mean distance are significantly smaller than the corresponding quantities of the local constant models.

In this part of the experiment, the local linear models produce the worst manifolds with respect to the mean absolute distance to the underlying smooth "Swiss Roll." Furthermore, they also exhibit by far the largest bias (mean signed distance), which is in accordance with our observations from the preceding experiments of this section.

type	degree	L_2 loss	Huber's loss
absolute distance Gaussian noise	local constant (UKR)	0.333 ± 0.022	0.253 ± 0.027
	local linear	0.462 ± 0.049	0.285 ± 0.048
	quad. w/o cross-terms	0.304 ± 0.034	0.212 ± 0.031
	local quadratic	0.279 ± 0.035	0.192 ± 0.027
signed distance Gaussian noise	local constant (UKR)	-0.018 ± 0.019	-0.033 ± 0.023
	local linear	-0.220 ± 0.036	-0.177 ± 0.060
	quad. w/o cross-terms	-0.028 ± 0.018	-0.004 ± 0.017
	local quadratic	-0.027 ± 0.019	-0.004 ± 0.018
absolute distance Laplacian noise	local constant (UKR)	0.309 ± 0.023	0.241 ± 0.030
	local linear	0.424 ± 0.054	0.265 ± 0.057
	quad. w/o cross-terms	0.277 ± 0.040	0.197 ± 0.032
	local quadratic	0.252 ± 0.037	0.184 ± 0.028
signed distance Laplacian noise	local constant (UKR)	-0.017 ± 0.025	-0.039 ± 0.045
	local linear	-0.203 ± 0.043	-0.159 ± 0.073
	quad. w/o cross-terms	-0.029 ± 0.021	-0.004 ± 0.020
	local quadratic	-0.024 ± 0.022	-0.005 ± 0.022

Table 5.12: Mean distance between the underlying smooth "Swiss Roll" and the UKR or ULPR reconstructions. All models were regularized by LKO-CV, $K = 3$.

Regarding the computational effort, we measured the time required for fine-tuning the models, that is, carrying out up to 1000 RPROP steps and gradient evaluations. The results, averaged across 200 trials[13], can be found in Table 5.13. Please note that we utilized the finite support Quartic kernel, and therefore the required computation time does not only depend on the particular degree of the underlying local models, but also on the sparseness of the matrix of basis functions.

The slightly higher computational effort of the quadratic models without the cross-term as compared to the full quadratic models can be explained from implementation details. In our MATLAB toolbox, or rather in the underlying subroutines written in C, the full quadratic model is hard coded for the case $q = 2$, while the code for the models without cross-terms can handle any value of q.

degree	time (seconds)
local constant (UKR)	30.4 ± 0.7
local linear	36.4 ± 2.2
quad. w/o cross-terms	48.3 ± 5.5
local quadratic	46.9 ± 5.0

Table 5.13: Required time for fine-tuning a UKR/ULPR model of a "Swiss Roll" dataset by up to 1000 RPROP steps. The results are averaged across 200 trials.

5.4.4. Discussion

In this section, we described how to generalize UKR to the family of Unsupervised Local Polynomial Regression algorithms, the members of which differ only in the choice of the polynomial degree of the local estimator. As was already known from the literature on classical (supervised) local polynomial regression, the Nadaraya-Watson estimator corresponds to a local constant fit and therefore exhibits an unfavorable behavior at the borders and – where curvature is present – also in the interior of the manifold. While local linear regression can remove the bias at the borders, only higher degrees (e.g., local quadratic regression) are useful to cancel out the curvature bias (cf. Hastie et al., 2001, Sec. 6.1).

In accordance to the way UKR was introduced as a counterpart of the Nadaraya-Watson estimator, we derived unsupervised variants of local linear and local quadratic regression, and we showed how especially an underlying local quadratic model can lead to improved manifolds. Most importantly, the smoothness vs. bias dilemma of standard UKR is not severe anymore, since the local quadratic models do not form simple convex combinations of the observed data vectors. This favorable property has been demonstrated clearly in the "noisy spiral" experiment, where the combination of local quadratic models and regularization by LKO-CV ($K = 5$) yielded manifolds that were very close to the true underlying spiral.

As a downside, the computational effort naturally increases with the degree of the local polynomials. For larger dimensionalities d of the data space, however, the effort

[13] The reported times correspond to the part of the experiment where we utilized LKO-CV and dealt with the datasets containing Laplacian noise.

is still dominated by the multiplication of the matrices $\mathbf{Y}$ and $\mathbf{B}$, which has complexity $O(dN^2)$ independently of the degree of the model. A more severe problem is a decreased numerical stability, resulting from the inversion of the matrix $\mathbf{C} = \bar{\mathbf{X}}\mathbf{W}\bar{\mathbf{X}}^T$ that is required in every evaluation of the regression function. This issue becomes most visible when a readily trained ULPR manifold is sampled, as for example in Fig. 5.27. At locations $\mathbf{x}$ where only few latent variables $\mathbf{x}_i$ contribute to the matrix $\mathbf{C}$ via the kernel $K(\mathbf{x} - \mathbf{x}_i)$, $\mathbf{C}$ may be ill-conditioned and not invertible without significant round-off errors, which then can cause strong deflections in the manifold.

Interestingly, the local linear models seem to suffer most from these numerical problems, which can be explained by the fact that a linear model can not handle curvature and therefore *must* sacrifice its smoothness (that is, a sufficient overlap of the kernels) for decreasing the reconstruction error. Indeed, our experiments suggest that unsupervised local linear regression is probably not worth considering further.

As we have stated in the beginning of Sec. 5.4, the variant of UKR we proposed here represents very recent work, and the examination and evaluation of its properties should certainly be carried further. In particular, we did not investigate models featuring a higher dimensional latent space, and we only presented experiments with synthetic data.

Nonetheless, the extension of UKR to an underlying local quadratic model is very appealing both from theoretical considerations and as demonstrated in the experiments we conducted. Besides the improvement of the numerical robustness, further interesting directions of future research on this extension are the combination of local quadratic models with landmark UKR (the gradient can already be found in appendix C.3), or a mixing strategy for homotopy-based optimization, where in early (coarse) homotopy steps a local constant model is fitted, and only in the final steps the more costly local quadratic regression is applied.

6. Conclusion

In this thesis, several new developments in the field of manifold learning have been described. Since each chapter or section containing research contributions has already been concluded with a corresponding discussion, we will only briefly repeat the most important points here.

As a first contribution, we presented an extension to the PSOM algorithm in chapter 3. In particular, we provided the PSOM with an explicit and elegant measure of the roughness of its regression manifold, and we showed how to make use of that roughness measure in order to learn PSOM mappings from noisy and incomplete training data. Our extension has important applications in robotics, where the PSOM has often been used in kinematic learning tasks. Consequently, we demonstrated the advantages of our method in the simulation of such a task.

In contrast to the original formulation, our extension also allows us to create PSOM mappings from data that is not organized on the nodes of a grid. Exploiting that property, we further proposed a PSOM variant for learning manifolds by alternating steps of data projection and parameter adaption, where the roughness measure can be utilized for controlling the model complexity.

As the central parts and contributions of this thesis, we presented the UKR algorithm and several fruitful extensions in chapters 4 and 5, respectively. Compared to other manifold learning algorithms, UKR first of all features an appealing approach to the model selection problem. Whereas other methods require the a priori specification of basis functions, grid layouts, or sample points, the structure of a UKR model is completely determined by the choice of a density kernel. Moreover, as we have seen in a number of experiments, that choice is of only little importance regarding the qualitative properties[1] of the model. Training a UKR model consists of minimizing a clearly defined objective function, which involves the error induced by reconstructing the observed data vectors from the model, with respect to the latent variables. In order to prevent an arbitrary interpolation solution, we have proposed several regularization variants. One of them – probably the strongest point of UKR – is to apply leave-one-out cross-validation (LOO-CV) without additional computational cost. Optionally, UKR manifolds can be fitted in an abstract feature space as implicitly defined by a Mercer kernel.

While the UKR objective function is highly non-convex, resulting in severe problems when using standard algorithms for local optimization, nonlinear spectral embedding methods can easily be utilized to generate initial sets of latent variables that are already close to a good solution. Moreover, multiple runs with different parameter adjustments of the spectral method can be compared with regard to the LOO-CV error of UKR, which greatly enhances the robustness of the combined algorithms and effectively cancels out the parameter specification for the spectral method. As an alternative, UKR models

[1] This is in stark contrast to e.g. Kernel PCA, where the choice of the Mercer kernel and its parameters heavily influence the results.

may be trained using a homotopy-based scheme, where the complexity is gradually increased in order to avoid shallow local minima.

In chapter 5, we presented several extensions to UKR, where sometimes we necessarily sheered away from the mission of posing as few parameters as possible, but we did so for a good reason. In particular, we demonstrated how to incorporate general loss functions within the UKR framework in Sec. 5.1. We concentrated on Huber's loss, which enhances the robustness of the UKR algorithm towards high noise levels and outliers in the data, and on the ϵ-insensitive loss, which can favorably be utilized to fit manifolds to data with a *known* inaccuracy. Concerning the latter, we proposed to first fit a possibly unsmooth manifold that is automatically regularized via LOO-CV – where information about the noise range cannot be exploited – and to afterwards smooth that manifold by using an ϵ-insensitive reconstruction error as a penalty function.

As a second extension, aimed at providing an intuitive regularization approach that produces smoother manifolds than LOO-CV, we introduced a leave-K-out cross-validation scheme that can be applied to UKR without increasing the computational cost. Generalizing the idea of LOO-CV, we proposed to reconstruct each data vector $\mathbf{y}_i$ without itself and its $K-1$ nearest neighbors, where we used fixed and pre-determined neighborhoods in the observed data space. We suggested a simple heuristics for the selection of the new parameter K, and we proposed an optional regularizer for tackling the problem of wriggly manifold borders, which may otherwise arise for rather large values of the leave-K-out parameter.

In section 5.3, we proposed a landmark variant of UKR that is primarily aimed at reducing the computational effort when fitting UKR models to large datasets. In particular, we suggested to generate a set of landmark points by application of a vector quantization or clustering algorithm, and to reconstruct the training data only from the landmark points. With the number n of those points, we introduced a new parameter that explicitly controls the computational effort, but also the shape complexity of the model. After having observed and discussed a rather large bias that results from choosing only a small number of landmark points, we proposed a further adaption step of the latter, for which we derived an explicit roughness measure of the UKR manifold in a similar way to our PSOM extensions in chapter 3. Since LOO-CV cannot be utilized to automatically regularize a landmark UKR model, the latter requires the selection of three parameters, which is rather unsatisfactory as compared to the original formulation. Still, experiments we conducted as a proof of concept recommend the landmark variant as a promising field of future research.

As the last contribution of this thesis, we described how to generalize UKR to Unsupervised Local Polynomial Regression (ULPR) in section 5.4. Inspired by the favorable properties of local linear and quadratic regression in the supervised case, we derived unsupervised variants of those algorithms in complete analogy to the way UKR was derived from the Nadaraya-Watson estimator, the latter corresponding to local constant regression. Consequently, the same regularization strategies (most notably LOO-CV) can be applied also for ULPR. We compared local constant (standard UKR), local linear, and local quadratic models in a number of experiments, where we observed that local linear models actually lead to worse results than standard UKR models, but that local quadratic models are highly advantageous. In contrast to standard UKR, local quadratic models can yield manifolds that are both smooth and show little or no bias,

which comes at the price of an (albeit only slightly) increased computational cost.

Most of the aforementioned extensions can be combined, and we actually investigated some possible combinations experimentally. In particular, in section 5.4.3 we fitted UKR and ULPR models to "noisy spiral" and "Swiss Roll" datasets, applying Huber's loss function and LKO-CV for regularization. With the UKR toolbox we developed for MATLAB, such combinations can be applied very easily.

In this thesis, most of the experimental evaluation has been carried out on synthetic "toy" datasets, and with the exception of the USPS digit classifier from Sec. 4.6.3 and 5.2.3, we did not present any real application of UKR and our extensions. By following the tradition of utilizing toy data, however, we were able to examine the properties of our methods in greater detail, for example by calculating the distance between a UKR model and the true underlying manifold from which the training data was sampled. In general, the underlying manifold is not known, and therefore it is much harder to judge the gain of new variants or algorithms. Nonetheless, for a broad acceptance of the UKR algorithm and its extensions, it will be necessary to demonstrate its usefulness in real applications. Indeed, first steps of utilizing UKR to improve grasping postures of a robotic hand have already been undertaken in collaboration with colleagues.

As another direction of future research, one could try to derive convergence bounds and learning rates similar to the works of Kégl et al. (2000) and Smola et al. (2001), which can provide further insight into the algorithm. On the other hand, it is always doubtful whether such an asymptotic theory is really relevant for the practical usage of an algorithm. The SOM, as an extreme example, is immensely popular despite its theoretical shortcomings, and gaining only a fraction of that popularity would be a huge success for UKR.

A. Mathematical notation

$\mathbb{R}$	field of real numbers
$\mathbb{R}^q$	space of q-dimensional real vectors
x	real number, element of $\mathbb{R}$
$f(\cdot)$	real-valued function
$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \dots \\ x_q \end{pmatrix}$	(column) vector with elements x_i
$\mathbf{f}(\cdot)$	vector-valued function
$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$	matrix with $m \times n$ elements $(\mathbf{A})_{ij} = a_{ij}$
$\mathrm{diag}(\mathbf{a})$	diagonal matrix with elements given by a_i
$\mathbf{A}^T$	transpose of the matrix $\mathbf{A}$, where $\left(\mathbf{A}^T\right)_{ij} = a_{ji}$
$\mathbf{a} \cdot \mathbf{b} = \mathbf{a}^T\mathbf{b}$	dot product $\sum_i a_i b_i$ of $\mathbf{a}$ and $\mathbf{b}$
$\langle \mathbf{a}, \mathbf{b} \rangle$	dot product between abstract vectors
$\mathbf{AB}$	matrix product, $(\mathbf{AB})_{ij} = \sum_k a_{ik} b_{kj}$
$\mathbf{A} * \mathbf{B}$	element-wise (Schur) product, $(\mathbf{A} * \mathbf{B})_{ij} = a_{ij} b_{ij}$
$\{\mathbf{x}_i\}$	set of vectors with index i
δ_{ij}	Kronecker symbol, $\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$
$\mathbf{I}_n$	identity matrix of size $n \times n$
$\mathbf{I}$	identity matrix, if size is clear from the context
$\mathbf{1}_n$	vector with n elements equal to 1
$\mathbf{1}$	same as above, if size is clear from the context
$\det(\mathbf{A})$	determinant of $\mathbf{A}$, product of eigenvalues
$\mathrm{tr}(\mathbf{A})$	trace of $\mathbf{A}$, sum $\sum_i a_{ii}$ of diagonal elements
$\delta(x)$	Dirac distribution, defined by $\int f(x)\delta(x)dx = f(0)$
$p(x)$	probability distribution of a random variable x
$\langle x \rangle$	expectation of a random variable x
$p(y\|x)$	conditional distribution of y given x
$\langle y\|x \rangle$	conditional expectation of y given x

Vector and matrix norms. The Euclidean norm of a vector $\mathbf{x} \in \mathbb{R}^q$ is given by

$$\|\mathbf{x}\| = \|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{q} x_i^2},$$

while the L_1 norm of that vector is defined as

$$\|\mathbf{x}\|_1 = \sum_i |x_i|.$$

The Frobenius norm of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ is given by

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2} = \sqrt{\mathrm{tr}(\mathbf{A}^T \mathbf{A})} = \sqrt{\mathrm{tr}(\mathbf{A}\mathbf{A}^T)}.$$

The matrix norm $\|\mathbf{A}\|$ equals the largest singular value of $\mathbf{A}$, with valid inequalities

$$\|\mathbf{AB}\| \leq \|\mathbf{A}\|\|\mathbf{B}\| \quad \text{and} \quad \|\mathbf{A}\| \leq \|\mathbf{A}\|_F.$$

B. UKR gradient and computational complexity

We write the UKR reconstruction error utilizing a general differentiable loss function $L(\cdot)$ (cf. Sec. 5.1.2) in the form

$$R_L(\mathbf{X}) = \frac{1}{N}\sum_j L\left(\mathbf{f}(\mathbf{x}_j;\mathbf{X}) - \mathbf{y}_j\right) = \frac{1}{N}\sum_j L(\mathbf{r}_j), \tag{B.1}$$

where the residuals $\mathbf{r}_j$ are given by

$$\mathbf{r}_j = \mathbf{Y}\mathbf{b}(\mathbf{x}_j;\mathbf{X}) - \mathbf{y}_j = \sum_{i=1}^{N} \mathbf{y}_i b_i(\mathbf{x}_j;\mathbf{X}) - \mathbf{y}_j. \tag{B.2}$$

The derivatives of the j-th residual $\mathbf{r}_j$ with respect to the latent variables can thus be expressed by

$$\frac{\partial \mathbf{r}_j}{\partial x_{mn}} = \sum_{k=1}^{N} \mathbf{y}_k \frac{\partial b_k(\mathbf{x}_j;\mathbf{X})}{\partial x_{mn}}, \tag{B.3}$$

where we used the notation $x_{mn} = (\mathbf{X})_{mn} = (\mathbf{x}_n)_m$. Then, introducing a $d \times N$ matrix $\mathbf{D}$ with the j-th column given by $\mathbf{d}_j = \nabla_{\mathbf{r}_j} L(\mathbf{r}_j)$, the gradient of (B.1) can be expressed by

$$\begin{aligned}
\frac{\partial R_L(\mathbf{X})}{\partial x_{mn}} &= \frac{1}{N}\sum_j \nabla_{\mathbf{r}_j} L(\mathbf{r}_j) \cdot \frac{\partial \mathbf{r}_j}{\partial x_{mn}} \\
&= \frac{1}{N}\sum_{i,j} \mathbf{d}_j \cdot \mathbf{y}_i \frac{\partial b_i(\mathbf{x}_j;\mathbf{X})}{\partial x_{mn}} \\
&= \frac{1}{N}\sum_{i,j} (\mathbf{Y}^T\mathbf{D})_{ij} \frac{\partial b_i(\mathbf{x}_j;\mathbf{X})}{\partial x_{mn}}.
\end{aligned} \tag{B.4}$$

For further computations, we restrict the choice of the density kernel to functions of the form

$$K(\mathbf{x}_i - \mathbf{x}_j) = F(\|\mathbf{x}_i - \mathbf{x}_j\|^2), \tag{B.5}$$

with derivatives given by

$$\frac{\partial K(\mathbf{x}_i - \mathbf{x}_j)}{\partial x_{mn}} = 2F'(\|\mathbf{x}_i - \mathbf{x}_j\|^2)(x_{mi} - x_{mj})(\delta_{in} - \delta_{jn}). \tag{B.6}$$

As a companion to the matrix of basis functions $\mathbf{B}(\mathbf{X})$ with elements

$$b_{ij} = b_i(\mathbf{x}_j;\mathbf{X}) = \frac{K(\mathbf{x}_i - \mathbf{x}_j)}{\sum_k K(\mathbf{x}_k - \mathbf{x}_j)}, \tag{B.7}$$

we now define a matrix $\mathbf{P}(\mathbf{X})$ with elements

$$p_{ij} = \frac{-2F'(\|\mathbf{x}_i - \mathbf{x}_j\|^2)}{\sum_k K(\mathbf{x}_k - \mathbf{x}_j)}. \tag{B.8}$$

Now, the partial derivatives of the basis functions can be expressed by

$$\begin{aligned}\frac{\partial b_{ij}}{\partial x_{mn}} &= \frac{\frac{\partial}{\partial x_{mn}} K(\mathbf{x}_i - \mathbf{x}_j)}{\sum_k K(\mathbf{x}_k - \mathbf{x}_j)} - \frac{K(\mathbf{x}_i - \mathbf{x}_j)}{(\sum_l K(\mathbf{x}_l - \mathbf{x}_j))^2} \sum_k \frac{\partial}{\partial x_{mn}} K(\mathbf{x}_k - \mathbf{x}_j) \\ &= p_{ij}\left[\delta_{in}x_{mj} - \delta_{in}x_{mi} - \delta_{jn}x_{mj} + \delta_{jn}x_{mi}\right] \\ &\quad -b_{ij}\left[\delta_{jn}\sum_k p_{kj}(x_{mk} - x_{mj}) - p_{nj}(x_{mn} - x_{mj})\right]. \end{aligned} \tag{B.9}$$

Inserting (B.9) into (B.4), eliminating the Kronecker symbols, and factoring out terms x_{m*} yields

$$\frac{\partial R_L(\mathbf{X})}{\partial x_{mn}} = \frac{1}{N}\sum_i (x_{mi} - x_{mn})(q_{ni} + q_{in}), \tag{B.10}$$

where we used the elements

$$q_{ni} = p_{ni}(\mathbf{Y}^T\mathbf{D})_{ni} - p_{ni}\sum_j (\mathbf{Y}^T\mathbf{D})_{ji} b_{ji} \tag{B.11}$$

of an $N \times N$-matrix $\mathbf{Q}$. By introducing $\mathbf{1}$ as the vector of N ones, we can therefore write the complete gradient in matrix form:

$$\frac{\partial R_L(\mathbf{X})}{\partial \mathbf{X}} = \frac{1}{N}\mathbf{X}\left\{\mathbf{Q} + \mathbf{Q}^T - \operatorname{diag}(\mathbf{1}^T[\mathbf{Q} + \mathbf{Q}^T])\right\}, \tag{B.12}$$

with the matrix

$$\mathbf{Q} = \mathbf{P} * \left\{\mathbf{Y}^T\mathbf{D} - \mathbf{1}\mathbf{1}^T[(\mathbf{Y}^T\mathbf{D}) * \mathbf{B}]\right\} \tag{B.13}$$

expressed by the help of the (element-wise) Schur product. The special case of $L(\cdot)$ being the squared L_2-norm yields $L(\mathbf{r}_j) = \mathbf{r}_j^T\mathbf{r}_j$ and thus $\mathbf{d}_j = 2\mathbf{r}_j$ or equivalently $\mathbf{D} = 2(\mathbf{YB} - \mathbf{Y})$, by which the gradient coincides with the original version published by Meinicke et al. (2005). Note that in this case $\mathbf{Y}^T\mathbf{D} = 2\mathbf{Y}^T\mathbf{Y}(\mathbf{B} - \mathbf{I}) = 2\mathbf{M}$ (cf. Eq. 4.50). For Huber's loss function (5.1), and with r_{ij} denoting the i-th component of the residual $\mathbf{r}_j$, the elements of the matrix $\mathbf{D}$ are given by

$$D_{ij} = (\mathbf{d}_j)_i = (\nabla L(\mathbf{r}_j))_i = \begin{cases} 1 & r_{ij} > \delta \\ -1 & r_{ij} < -\delta \\ \frac{1}{\delta} r_{ij} & \text{otherwise,} \end{cases} \tag{B.14}$$

whereas for the squared ϵ-insensitive loss (5.2) in the form $L(\mathbf{r}) = \sum_i l_\epsilon^2(r_i)$ these elements are given by

$$D_{ij} = \begin{cases} 2(r_{ij} - \epsilon) & r_{ij} > \epsilon \\ 2(\epsilon - r_{ij}) & r_{ij} < -\epsilon \\ 0 & \text{otherwise.} \end{cases} \tag{B.15}$$

For calculating the gradient of the cross-validated reconstruction error (4.34) or (5.6), one has to replace all occurences of the form $K(\mathbf{x}_i - \mathbf{x}_i)$ and $f'(\|\mathbf{x}_i - \mathbf{x}_i\|)$ with 0, effectively zero-ing the diagonals of $\mathbf{B}$ and $\mathbf{P}$.

Regarding the computational complexity, the most expensive parts of a gradient evaluation are the matrix-matrix multiplications $\mathbf{Y}^T\mathbf{D}$ and $\mathbf{YB}$ (used for calculating the residuals), which are $O(dN^2)$ operations. However, when using a finite support kernel function (e.g. Quartic), the matrices $\mathbf{B}$ and $\mathbf{P}$ become sparse, which can yield drastic speed-ups when utilizing a sparse linear algebra package for calculating $\mathbf{YB}$. Furthermore, the product $\mathbf{Y}^T\mathbf{D}$ is only used within Schur products with $\mathbf{B}$ and $\mathbf{P}$, so many of its elements are irrelevant and thus do not have to be computed. Another possible speed-up arises from utilizing the ϵ-insensitive loss function, through which also the matrix $\mathbf{D}$ becomes sparse.

C. Derivative calculations for Unsupervised Local Polynomial Regression

C.1. Gradient of the reconstruction error

At first we write the ULPR regression function in a way similar to the standard UKR function, that is, we "hide" everything but the data matrix $\mathbf{Y}$ inside a vector $\mathbf{b}$ of basis functions:

$$\mathbf{f}(\mathbf{x};\mathbf{X}) = \mathbf{m}_\mathbf{x} = \mathbf{Y}\mathbf{W}\bar{\mathbf{X}}^T\mathbf{C}^{-1}\mathbf{e}_0 = \mathbf{Y}\mathbf{b}(\mathbf{x};\mathbf{X}). \tag{C.1}$$

This allows us to write the reconstruction error in the familiar form

$$R(\mathbf{X}) = \frac{1}{N}\sum_{j=1}^{N}\|\mathbf{y}_j - \mathbf{f}(\mathbf{x}_j;\mathbf{X})\|^2 = \frac{1}{N}\|\mathbf{Y} - \mathbf{Y}\mathbf{B}(\mathbf{X})\|_F^2. \tag{C.2}$$

Now we need expressions for the columns $\mathbf{b}(\mathbf{x}_j;\mathbf{X})$ of $\mathbf{B}(\mathbf{X})$ and its derivatives, for which we introduce some shortcut notation:

$$\mathbf{b}(\mathbf{x}_j;\mathbf{X}) = \mathbf{W}^j\bar{\mathbf{X}}^{j^T}\mathbf{C}^{j^{-1}}\mathbf{e}_0 = \mathbf{V}^j\mathbf{C}^{j^{-1}}\mathbf{e}_0. \tag{C.3}$$

Here and in the following, an upper index j denotes the dependence of a matrix or vector on the position $\mathbf{x}_j$ where we evaluate the regression function. $\mathbf{W}^j$ is a diagonal matrix with the entries

$$(\mathbf{W}^j)_{il} = \delta_{il}K(\mathbf{x}_i - \mathbf{x}_j) = \delta_{il}K_i^j. \tag{C.4}$$

For the entries of the design matrix $\bar{\mathbf{X}}^j$, we introduce the notation $(\bar{\mathbf{X}}^j)_{\mu i} = \xi_{\mu i}^j$. In the local linear case, we have

$$\xi_{\mu i}^j = \begin{pmatrix} 1 \\ \mathbf{x}_i - \mathbf{x}_j \end{pmatrix}_\mu$$

and μ ranges from 0 to q, whereas in the quadratic case ($q = 1$), we have

$$\xi_{\mu i}^j = \begin{pmatrix} 1 \\ x_i - x_j \\ (x_i - x_j)^2 \end{pmatrix}_\mu$$

and μ ranges from 0 to 2. For the local constant case (Nadaraya-Watson), we simply have $\xi_i^j = 1$, which should yield the same gradient as already known from standard UKR. We can now express the elements of the matrices $\mathbf{V}^j$ and $\mathbf{C}^j$ by

$$V_{i\nu}^j = (\mathbf{V}^j)_{i\nu} = (\mathbf{W}^j\bar{\mathbf{X}}^{j^T})_{i\nu} = \sum_{l=1}^{N}(\mathbf{W}^j)_{il}(\bar{\mathbf{X}}^{j^T})_{l\nu} = \sum_{l=1}^{N}\delta_{il}K_i^j\xi_{\nu l}^j = K_i^j\xi_{\nu i}^j \tag{C.5}$$

and

$$C^j_{\mu\nu} = (\mathbf{C}^j)_{\mu\nu} = (\bar{\mathbf{X}}^j \mathbf{W}^j \bar{\mathbf{X}}^{j^T})_{\mu\nu} = (\bar{\mathbf{X}}^j \mathbf{V}^j)_{\mu\nu} = \sum_{i=1}^{N} \xi^j_{\mu i} K^j_i \xi^j_{\nu i}. \tag{C.6}$$

Later we will use the notation $\Gamma^j_{\mu\nu}$ for the elements of the inverse matrix $\mathbf{C}^{j^{-1}}$. We now start with the actual calculation of the gradient, where we stay with the general case as long as possible. Summations over latin indices are understood to range from 1 to N.

$$\frac{\partial R(\mathbf{X})}{\partial x_{mn}} = \frac{2}{N} \sum_{i,j} \left(\mathbf{Y}^T (\mathbf{Y} - \mathbf{YB}(\mathbf{X})\right)_{ij} \frac{\partial b_{ij}(\mathbf{X})}{\partial x_{mn}} = \frac{2}{N} \sum_{i,j} M_{ij} \frac{\partial b_{ij}}{\partial x_{mn}} \tag{C.7}$$

$$b_{ij} = b_{ij}(\mathbf{X}) = (\mathbf{b}(\mathbf{x}_j; \mathbf{X}))_i = \left(\mathbf{V}^j \mathbf{C}^{j^{-1}} \mathbf{e}_0\right)_i \tag{C.8}$$

$$\begin{aligned}
\frac{\partial b_{ij}}{\partial x_{mn}} &= \frac{\partial}{\partial x_{mn}} \left(\mathbf{V}^j \mathbf{C}^{j^{-1}} \mathbf{e}_0\right)_i \\
&= \left(\left[\frac{\partial \mathbf{V}^j}{\partial x_{mn}} \mathbf{C}^{j^{-1}} + \mathbf{V}^j \frac{\partial \mathbf{C}^{j^{-1}}}{\partial x_{mn}}\right] \mathbf{e}_0\right)_i \\
&= \left(\left[\frac{\partial \mathbf{V}^j}{\partial x_{mn}} \mathbf{C}^{j^{-1}} - \mathbf{V}^j \mathbf{C}^{j^{-1}} \frac{\partial \mathbf{C}^j}{\partial x_{mn}} \mathbf{C}^{j^{-1}}\right] \mathbf{e}_0\right)_i \\
&= \left(\left[\frac{\partial \mathbf{V}^j}{\partial x_{mn}} - \mathbf{V}^j \mathbf{C}^{j^{-1}} \frac{\partial \mathbf{C}^j}{\partial x_{mn}}\right] \mathbf{C}^{j^{-1}} \mathbf{e}_0\right)_i \\
&= \sum_\nu \left[\frac{\partial \mathbf{V}^j}{\partial x_{mn}} - \mathbf{V}^j \mathbf{C}^{j^{-1}} \frac{\partial \mathbf{C}^j}{\partial x_{mn}}\right]_{i\nu} \Gamma^j_{\nu 0} \\
&= \sum_\nu \left[\frac{\partial V^j_{i\nu}}{\partial x_{mn}} - \sum_{\alpha,\mu} V^j_{i\alpha} \Gamma^j_{\alpha\mu} \frac{\partial C^j_{\mu\nu}}{\partial x_{mn}}\right] \Gamma^j_{\nu 0}
\end{aligned} \tag{C.9}$$

$$\frac{\partial K^j_i}{\partial x_{mn}} = \underbrace{2K'(\|\mathbf{x}_i - \mathbf{x}_j\|^2)}_{-P^j_i} (x_{mi} - x_{mj})(\delta_{in} - \delta_{jn}) \tag{C.10}$$

A general form of the derivatives of the variables $\xi^j_{\nu i}$ can be given as

$$\frac{\partial \xi^j_{\nu i}}{\partial x_{mn}} = \chi^{ij}_{m\nu} (\delta_{in} - \delta_{jn}) \quad \text{with} \quad \chi^{ij}_{m0} = 0. \tag{C.11}$$

$$\begin{aligned}
\frac{\partial V^j_{i\nu}}{\partial x_{mn}} &= \frac{\partial K^j_i}{\partial x_{mn}} \xi^j_{\nu i} + K^j_i \frac{\partial \xi^j_{\nu i}}{\partial x_{mn}} \\
&= \left[-P^j_i (x_{mi} - x_{mj}) \xi^j_{\nu i} + K^j_i \chi^{ij}_{m\nu}\right] (\delta_{in} - \delta_{jn})
\end{aligned} \tag{C.12}$$

$$\begin{aligned}
\frac{\partial C^j_{\mu\nu}}{\partial x_{mn}} &= \sum_i \left[\frac{\partial \xi^j_{\mu i}}{\partial x_{mn}} K^j_i \xi^j_{\nu i} + \xi^j_{\mu i} \frac{\partial K^j_i}{\partial x_{mn}} \xi^j_{\nu i} + \xi^j_{\mu i} K^j_i \frac{\partial \xi^j_{\nu i}}{\partial x_{mn}}\right] \\
&= \sum_i \left[\chi^{ij}_{m\mu} K^j_i \xi^j_{\nu i} - \xi^j_{\mu i} P^j_i (x_{mi} - x_{mj}) \xi^j_{\nu i} + \xi^j_{\mu i} K^j_i \chi^{ij}_{m\nu}\right] (\delta_{in} - \delta_{jn}).
\end{aligned} \tag{C.13}$$

We now use an additional definition (which is symmetric in μ and ν)

$$\Lambda^{ij}_{m\mu\nu} = \chi^{ij}_{m\mu} K^j_i \xi^j_{\nu i} + \xi^j_{\mu i} K^j_i \chi^{ij}_{m\nu} = K^j_i (\chi^{ij}_{m\mu} \xi^j_{\nu i} + \chi^{ij}_{m\nu} \xi^j_{\mu i}), \tag{C.14}$$

through which we can write

$$\begin{aligned} \frac{\partial C^j_{\mu\nu}}{\partial x_{mn}} &= \sum_i \left[\Lambda^{ij}_{m\mu\nu} - \xi^j_{\mu i} P^j_i (x_{mi} - x_{mj}) \xi^j_{\nu i} \right] (\delta_{in} - \delta_{jn}) \\ &= \Lambda^{nj}_{m\mu\nu} - \xi^j_{\mu n} P^j_n (x_{mn} - x_{mj}) \xi^j_{\nu n} \\ &\quad - \delta_{jn} \sum_k \left[\Lambda^{kj}_{m\mu\nu} - \xi^j_{\mu k} P^j_k (x_{mk} - x_{mj}) \xi^j_{\nu k} \right]. \end{aligned} \tag{C.15}$$

From now on, we use Einstein's summation convention, that is, we automatically sum over double indices, but only if they are greek letters. Therefore, we can express (C.9) more compactly as

$$\frac{\partial b_{ij}}{\partial x_{mn}} = \frac{\partial V^j_{i\nu}}{\partial x_{mn}} \Gamma^j_{\nu 0} - V^j_{i\alpha} \Gamma^j_{\alpha\mu} \frac{\partial C^j_{\mu\nu}}{\partial x_{mn}} \Gamma^j_{\nu 0} \tag{C.16}$$

with an automatic summation over μ, ν and α. Inserting (C.12) and (C.15) yields

$$\begin{aligned} \frac{\partial b_{ij}}{\partial x_{mn}} &= \left[-P^j_i (x_{mi} - x_{mj}) \xi^j_{\nu i} \Gamma^j_{\nu 0} + K^j_i \chi^{ij}_{m\nu} \Gamma^j_{\nu 0} \right] (\delta_{in} - \delta_{jn}) \\ &\quad - V^j_{i\alpha} \Gamma^j_{\alpha\mu} \Lambda^{nj}_{m\mu\nu} \Gamma^j_{\nu 0} + V^j_{i\alpha} \Gamma^j_{\alpha\mu} \xi^j_{\mu n} P^j_n (x_{mn} - x_{mj}) \xi^j_{\nu n} \Gamma^j_{\nu 0} \\ &\quad + \delta_{jn} \sum_k \left[V^j_{i\alpha} \Gamma^j_{\alpha\mu} \Lambda^{kj}_{m\mu\nu} \Gamma^j_{\nu 0} - V^j_{i\alpha} \Gamma^j_{\alpha\mu} \xi^j_{\mu k} P^j_k (x_{mk} - x_{mj}) \xi^j_{\nu k} \Gamma^j_{\nu 0} \right]. \end{aligned} \tag{C.17}$$

This term is now inserted into (C.7):

$$\begin{aligned} \frac{N}{2} \frac{\partial R(\mathbf{X})}{\partial x_{mn}} &= \sum_{i,j} M_{ij} \frac{\partial b_{ij}}{\partial x_{mn}} \\ &= \sum_{i,j} M_{ij} \left[-P^j_i (x_{mi} - x_{mj}) \xi^j_{\nu i} \Gamma^j_{\nu 0} + K^j_i \chi^{ij}_{m\nu} \Gamma^j_{\nu 0} \right] \delta_{in} \\ &\quad + \sum_{i,j} M_{ij} \left[P^j_i (x_{mi} - x_{mj}) \xi^j_{\nu i} \Gamma^j_{\nu 0} - K^j_i \chi^{ij}_{m\nu} \Gamma^j_{\nu 0} \right] \delta_{jn} \\ &\quad - \sum_{i,j} M_{ij} V^j_{i\alpha} \Gamma^j_{\alpha\mu} \Lambda^{nj}_{m\mu\nu} \Gamma^j_{\nu 0} \\ &\quad + \sum_{i,j} M_{ij} V^j_{i\alpha} \Gamma^j_{\alpha\mu} \xi^j_{\mu n} P^j_n (x_{mn} - x_{mj}) \xi^j_{\nu n} \Gamma^j_{\nu 0} \\ &\quad + \sum_{i,j} M_{ij} \delta_{jn} \sum_k V^j_{i\alpha} \Gamma^j_{\alpha\mu} \Lambda^{kj}_{m\mu\nu} \Gamma^j_{\nu 0} \\ &\quad - \sum_{i,j,k} M_{ij} \delta_{jn} V^j_{i\alpha} \Gamma^j_{\alpha\mu} \xi^j_{\mu k} P^j_k (x_{mk} - x_{mj}) \xi^j_{\nu k} \Gamma^j_{\nu 0}. \end{aligned} \tag{C.18}$$

A summation over the Kronecker symbols yields

$$\begin{aligned}\frac{N}{2}\frac{\partial R(\mathbf{X})}{\partial x_{mn}} &= \sum_j M_{nj}\left[P_n^j(x_{mj}-x_{mn})\xi_{\nu n}^j\Gamma_{\nu 0}^j + K_n^j\chi_{m\nu}^{nj}\Gamma_{\nu 0}^j\right] \\ &+\sum_i M_{in}\left[P_i^n(x_{mi}-x_{mn})\xi_{\nu i}^n\Gamma_{\nu 0}^n - K_i^n\chi_{m\nu}^{in}\Gamma_{\nu 0}^n\right] \\ &-\sum_{i,j} M_{ij}V_{i\alpha}^j\Gamma_{\alpha\mu}^j\Lambda_{m\mu\nu}^{nj}\Gamma_{\nu 0}^j \\ &+\sum_{i,j} M_{ij}V_{i\alpha}^j\Gamma_{\alpha\mu}^j\xi_{\mu n}^j P_n^j(x_{mn}-x_{mj})\xi_{\nu n}^j\Gamma_{\nu 0}^j \\ &+\sum_{i,k} M_{in}V_{i\alpha}^n\Gamma_{\alpha\mu}^n\Lambda_{m\mu\nu}^{kn}\Gamma_{\nu 0}^n \\ &-\sum_{i,k} M_{in}V_{i\alpha}^n\Gamma_{\alpha\mu}^n\xi_{\mu k}^n P_k^n(x_{mk}-x_{mn})\xi_{\nu k}^n\Gamma_{\nu 0}^n. \end{aligned} \tag{C.19}$$

Now we introduce the shortcut notation $\Psi_\alpha^n = \sum_i M_{in}V_{i\alpha}^n$ and change all remaining summation indices to k:

$$\begin{aligned}\frac{N}{2}\frac{\partial R(\mathbf{X})}{\partial x_{mn}} &= \sum_k M_{nk}\left[P_n^k(x_{mk}-x_{mn})\xi_{\nu n}^k\Gamma_{\nu 0}^k + K_n^k\chi_{m\nu}^{nk}\Gamma_{\nu 0}^k\right] \\ &+\sum_k M_{kn}\left[P_k^n(x_{mk}-x_{mn})\xi_{\nu k}^n\Gamma_{\nu 0}^n - K_k^n\chi_{m\nu}^{kn}\Gamma_{\nu 0}^n\right] \\ &-\sum_k \Psi_\alpha^k\Gamma_{\alpha\mu}^k\Lambda_{m\mu\nu}^{nk}\Gamma_{\nu 0}^k \\ &+\sum_k \Psi_\alpha^k\Gamma_{\alpha\mu}^k\xi_{\mu n}^k P_n^k(x_{mn}-x_{mk})\xi_{\nu n}^k\Gamma_{\nu 0}^k \\ &+\sum_k \Psi_\alpha^n\Gamma_{\alpha\mu}^n\Lambda_{m\mu\nu}^{kn}\Gamma_{\nu 0}^n \\ &-\sum_k \Psi_\alpha^n\Gamma_{\alpha\mu}^n\xi_{\mu k}^n P_k^n(x_{mk}-x_{mn})\xi_{\nu k}^n\Gamma_{\nu 0}^n \end{aligned} \tag{C.20}$$

In the next step, we factor out $(x_{mk}-x_{mn})$ and sort the remaining terms according to the occurence of χ and Λ.

$$\begin{aligned}\frac{N}{2}\frac{\partial R(\mathbf{X})}{\partial x_{mn}} &= \sum_k (x_{mk}-x_{mn})\Big[M_{nk}P_n^k\xi_{\nu n}^k\Gamma_{\nu 0}^k + M_{kn}P_k^n\xi_{\nu k}^n\Gamma_{\nu 0}^n \\ &\qquad - \Psi_\alpha^k\Gamma_{\alpha\mu}^k\xi_{\mu n}^k P_n^k\xi_{\nu n}^k\Gamma_{\nu 0}^k - \Psi_\alpha^n\Gamma_{\alpha\mu}^n\xi_{\mu k}^n P_k^n\xi_{\nu k}^n\Gamma_{\nu 0}^n\Big] \\ &+\sum_k M_{nk}K_n^k\chi_{m\nu}^{nk}\Gamma_{\nu 0}^k - \sum_k M_{kn}K_k^n\chi_{m\nu}^{kn}\Gamma_{\nu 0}^n \\ &-\sum_k \Psi_\alpha^k\Gamma_{\alpha\mu}^k\Lambda_{m\mu\nu}^{nk}\Gamma_{\nu 0}^k + \sum_k \Psi_\alpha^n\Gamma_{\alpha\mu}^n\Lambda_{m\mu\nu}^{kn}\Gamma_{\nu 0}^n. \end{aligned} \tag{C.21}$$

We note that the term inside the square bracket is symmetric in $k \leftrightarrow n$, while the rest of the expression is anti-symmetric. Thus, we can write the gradient of general

Unsupervised Local Polynomial Regression as

$$\frac{\partial R(\mathbf{X})}{\partial x_{mn}} = \frac{2}{N} \sum_k \left[(x_{mk} - x_{mn})(Q_k^n + Q_n^k) + T_{mn}^k - T_{mk}^n \right], \tag{C.22}$$

where we used the definitions

$$Q_k^n = M_{kn} P_k^n \xi_{\nu k}^n \Gamma_{\nu 0}^n - \Psi_\alpha^n \Gamma_{\alpha\mu}^n \xi_{\mu k}^n P_k^n \xi_{\nu k}^n \Gamma_{\nu 0}^n \tag{C.23}$$

and

$$T_{mk}^n = M_{kn} K_k^n \chi_{m\nu}^{kn} \Gamma_{\nu 0}^n - \Psi_\alpha^n \Gamma_{\alpha\mu}^n \Lambda_{m\mu\nu}^{kn} \Gamma_{\nu 0}^n. \tag{C.24}$$

C.1.1. Local Constant Estimate

Please note that in this case, the resulting gradient must be identical to the one we know from standard UKR, and we present the calculation as a verification of our general form. The index $m = 1$, whereas the greek indexes are 0. We have

$$\xi_i^j = 1 \quad \text{and} \quad \chi^{ij} = \Lambda^{ij} = 0 \tag{C.25}$$

as well as

$$\Psi_\alpha^j = \sum_i M_{ij} V_{i\alpha}^j = \sum_i M_{ij} K_i^j \xi_{\alpha i}^j = \sum_i M_{ij} K_i^j. \tag{C.26}$$

The matrix $\mathbf{C}^j$ and its inverse are just scalars, given by

$$C^j = \sum_l K_l^j =: S^j \quad \text{and} \quad \Gamma^j = \frac{1}{S^j}.$$

Inserting these values into (C.23) and (C.24) yields

$$Q_k^n = M_{kn} P_k^n \frac{1}{S^n} - \sum_i M_{in} K_i^n \frac{1}{S^n} P_k^n \frac{1}{S^n} \tag{C.27}$$

and

$$T_{mk}^n = 0. \tag{C.28}$$

If we now identify

$$\frac{K_i^j}{S^j} = \frac{K_i^j}{\sum_l K_i^j} = b_{ij} \quad \text{and} \quad \frac{P_i^j}{\sum_j K_l^j} = p_{ij}, \tag{C.29}$$

we get the form (cf. Eq. B.11)

$$Q_k^n = p_{kn} \left[M_{kn} - \sum_i M_{in} b_{in} \right] = q_{kn} \tag{C.30}$$

which we already know from standard UKR. Please note that a general loss function can be utilized just as with standard UKR, if only the matrix elements M_{ij} are changed as demonstrated in appendix B.

C.1.2. Local Linear Estimate

In this case, the entries of the design matrix are given by

$$\xi^j_{\mu i} = \begin{pmatrix} 1 \\ \mathbf{x}_i - \mathbf{x}_j \end{pmatrix}_\mu \tag{C.31}$$

and the greek indexes range from 0 to q. Since we defined χ by

$$\frac{\partial \xi^j_{\mu i}}{\partial x_{mn}} = \chi^{ij}_{m\mu}(\delta_{in} - \delta_{jn}), \tag{C.32}$$

we have

$$\chi^{ij}_{m\mu} = \delta_{m\mu} \tag{C.33}$$

and thus

$$\Lambda^{ij}_{m\mu\nu} = K^j_i(\delta_{m\mu}\xi^j_{\nu i} + \delta_{m\nu}\xi^j_{\mu i}). \tag{C.34}$$

Insertion into (C.24) yields

$$\begin{aligned} T^n_{mk} &= M_{kn}K^n_k\chi^{kn}_{m\nu}\Gamma^n_{\nu 0} - \Psi^n_\alpha\Gamma^n_{\alpha\mu}\Lambda^{kn}_{m\mu\nu}\Gamma^n_{\nu 0} \\ &= M_{kn}K^n_k\delta_{m\nu}\Gamma^n_{\nu 0} - \Psi^n_\alpha\Gamma^n_{\alpha\mu}K^n_k(\delta_{m\mu}\xi^n_{\nu k} + \delta_{m\nu}\xi^n_{\mu k})\Gamma^n_{\nu 0} \\ &= M_{kn}K^n_k\Gamma^n_{m0} - \Psi^n_\alpha\Gamma^n_{\alpha m}K^n_k\xi^n_{\nu k}\Gamma^n_{\nu 0} - \Psi^n_\alpha\Gamma^n_{\alpha\mu}K^n_k\xi^n_{\mu k}\Gamma^n_{m0}. \end{aligned} \tag{C.35}$$

We now define the abbreviation

$$U^n_{\nu k} = \Gamma^n_{\mu\nu}\xi^n_{\mu k} = \Gamma^n_{\nu\mu}\xi^n_{\mu k}, \tag{C.36}$$

by which we can write

$$T^n_{mk} = M_{kn}K^n_k\Gamma^n_{m0} - \Psi^n_\alpha\Gamma^n_{\alpha m}K^n_kU^n_{0k} - \Psi^n_\alpha U^n_{\alpha k}K^n_k\Gamma^n_{m0}. \tag{C.37}$$

Using the definition of U in (C.23) yields

$$Q^n_k = M_{kn}P^n_kU^n_{0k} - \Psi^n_\alpha U^n_{\alpha k}P^n_kU^n_{0k} = (M_{kn} - \Psi^n_\alpha U^n_{\alpha k})P^n_kU^n_{0k}. \tag{C.38}$$

C.1.3. Local Quadratic Estimate (w/o cross-terms)

In this section, we use a local quadratic estimate without cross-terms, that is we set

$$\xi^j_{\mu i} = \begin{cases} 1 & \mu = 0 \\ x_{\mu i} - x_{\mu j} & \mu = 1, 2 \dots q \\ (x_{(\mu-q)i} - x_{(\mu-q)j})^2 & \mu = q+1, q+2 \dots 2q. \end{cases} \tag{C.39}$$

The greek indexes range from 0 to $2q$. Since we defined χ by

$$\frac{\partial \xi^j_{\mu i}}{\partial x_{mn}} = \chi^{ij}_{m\mu}(\delta_{in} - \delta_{jn}), \tag{C.40}$$

we have

$$\chi^{ij}_{m\mu} = \delta_{m\mu} + 2(x_{mi} - x_{mj})\delta_{m(\mu-q)} = \delta_{m\mu} + 2\xi^j_{mi}\delta_{m(\mu-q)} \tag{C.41}$$

and thus

$$\Lambda^{ij}_{m\mu\nu} = K^j_i \left[\left(\delta_{m\mu} + 2\xi^j_{mi}\delta_{m(\mu-q)} \right) \xi^j_{\nu i} + \left(\delta_{m\nu} + 2\xi^j_{mi}\delta_{m(\nu-q)} \right) \xi^j_{\mu i} \right]. \tag{C.42}$$

Insertion into (C.24) yields

$$\begin{aligned}
T^n_{mk} &= M_{kn}K^n_k \chi^{kn}_{m\nu}\Gamma^n_{\nu 0} - \Psi^n_\alpha \Gamma^n_{\alpha\mu}\Lambda^{kn}_{m\mu\nu}\Gamma^n_{\nu 0} \\
&= M_{kn}K^n_k \left(\delta_{m\nu} + 2\xi^n_{mk}\delta_{m(\nu-q)} \right)\Gamma^n_{\nu 0} \\
&\quad -\Psi^n_\alpha \Gamma^n_{\alpha\mu} K^n_k \left(\delta_{m\mu} + 2\xi^n_{mk}\delta_{m(\mu-q)} \right)\xi^n_{\nu k}\Gamma^n_{\nu 0} \\
&\quad -\Psi^n_\alpha \Gamma^n_{\alpha\mu} K^n_k \left(\delta_{m\nu} + 2\xi^n_{mk}\delta_{m(\nu-q)} \right)\xi^n_{\mu k}\Gamma^n_{\nu 0} \\
&= M_{kn}K^n_k \left(\Gamma^n_{m0} + 2\xi^n_{mk}\Gamma^n_{(m+q)0} \right) \\
&\quad -\Psi^n_\alpha \left(\Gamma^n_{\alpha m} + 2\xi^n_{mk}\Gamma^n_{\alpha(m+q)} \right) K^n_k \xi^n_{\nu k}\Gamma^n_{\nu 0} \\
&\quad -\Psi^n_\alpha \Gamma^n_{\alpha\mu}\xi^n_{\mu k} K^n_k \left(\Gamma^n_{m0} + 2\xi^n_{mk}\Gamma^n_{(m+q)0} \right) \\
&= M_{kn}K^n_k \left(\Gamma^n_{m0} + 2\xi^n_{mk}\Gamma^n_{(m+q)0} \right) \\
&\quad -\left(\Psi^n_\alpha \Gamma^n_{\alpha m} + 2\xi^n_{mk}\Psi^n_\alpha \Gamma^n_{\alpha(m+q)} \right) K^n_k U^n_{0k} \\
&\quad -\Psi^n_\alpha U^n_{\alpha k} K^n_k \left(\Gamma^n_{m0} + 2\xi^n_{mk}\Gamma^n_{(m+q)0} \right) \\
&= \left(\Gamma^n_{m0} + 2\xi^n_{mk}\Gamma^n_{(m+q)0} \right)\left(M_{kn}K^n_k - \Psi^n_\alpha U^n_{\alpha k} K^n_k \right) \\
&\quad -\left(\Psi^n_\alpha \Gamma^n_{\alpha m} + 2\xi^n_{mk}\Psi^n_\alpha \Gamma^n_{\alpha(m+q)} \right) K^n_k U^n_{0k},
\end{aligned} \tag{C.43}$$

where we used our previous definition $U^n_{\nu k} = \Gamma^n_{\mu\nu}\xi^n_{\mu k}$. The elements Q^n_k have the same form as in the local linear case.

C.1.4. Local Quadratic Estimate ($q = 2$)

Here, we use a local quadratic estimate with cross-terms[1], but restrict ourselves to the (important) case $q = 2$. The elements of the design matrix are given by

$$\xi^j_{\mu i} = \begin{pmatrix} 1 \\ x_{1i} - x_{1j} \\ x_{2i} - x_{2j} \\ (x_{1i} - x_{1j})^2 \\ (x_{2i} - x_{2j})^2 \\ (x_{1i} - x_{1j})(x_{2i} - x_{2j}) \end{pmatrix}_\mu \tag{C.44}$$

with greek indexes ranging from 0 to 5. Here we have

$$\chi^{ij}_{1\mu} = \delta_{1\mu} + 2\xi^j_{1i}\delta_{3\mu} + 2\xi^j_{2i}\delta_{5\mu} \tag{C.45}$$

$$\chi^{ij}_{2\mu} = \delta_{2\mu} + 2\xi^j_{2i}\delta_{4\mu} + 2\xi^j_{1i}\delta_{5\mu} \tag{C.46}$$

[1] Please note that for $q = 1$ there are no cross-terms, so the results from the preceding sections already describes the complete model.

and thus

$$\Lambda^{ij}_{1\mu\nu} = K^j_i \left[\left(\delta_{1\mu} + 2\xi^j_{1i}\delta_{3\mu} + \xi^j_{2i}\delta_{5\mu} \right) \xi^j_{\nu i} + \left(\delta_{1\nu} + 2\xi^j_{1i}\delta_{3\nu} + \xi^j_{2i}\delta_{5\nu} \right) \xi^j_{\mu i} \right]. \tag{C.47}$$

Insertion into (C.24) yields

$$\begin{aligned}
T^n_{1k} &= M_{kn}K^n_k \chi^{kn}_{1\nu}\Gamma^n_{\nu 0} - \Psi^n_\alpha \Gamma^n_{\alpha\mu}\Lambda^{kn}_{1\mu\nu}\Gamma^n_{\nu 0} \\
&= M_{kn}K^n_k \left(\delta_{1\nu} + 2\xi^n_{1k}\delta_{3\nu} + \xi^n_{2k}\delta_{5\nu}\right)\Gamma^n_{\nu 0} \\
&\quad -\Psi^n_\alpha \Gamma^n_{\alpha\mu} K^n_k \left(\delta_{1\mu} + 2\xi^n_{1k}\delta_{3\mu} + \xi^n_{2k}\delta_{5\mu}\right)\xi^n_{\nu k}\Gamma^n_{\nu 0} \\
&\quad -\Psi^n_\alpha \Gamma^n_{\alpha\mu} K^n_k \left(\delta_{1\nu} + 2\xi^n_{1k}\delta_{3\nu} + \xi^n_{2k}\delta_{5\nu}\right)\xi^n_{\mu k}\Gamma^n_{\nu 0} \\
&= M_{kn}K^n_k \left(\Gamma^n_{10} + 2\xi^n_{1k}\Gamma^n_{30} + \xi^n_{2k}\Gamma^n_{50}\right) \\
&\quad -\Psi^n_\alpha \left(\Gamma^n_{\alpha 1} + 2\xi^n_{1k}\Gamma^n_{\alpha 3} + \xi^n_{2k}\Gamma^n_{\alpha 5}\right) K^n_k \xi^n_{\nu k}\Gamma^n_{\nu 0} \\
&\quad -\Psi^n_\alpha \Gamma^n_{\alpha\mu}\xi^n_{\mu k} K^n_k \left(\Gamma^n_{10} + 2\xi^n_{1k}\Gamma^n_{30} + \xi^n_{2k}\Gamma^n_{50}\right) \\
&= \left(\Gamma^n_{10} + 2\xi^n_{1k}\Gamma^n_{30} + \xi^n_{2k}\Gamma^n_{50}\right)\left(M_{kn}K^n_k - \Psi^n_\alpha U^n_{\alpha k}K^n_k\right) \\
&\quad -\left(\Psi^n_\alpha\Gamma^n_{\alpha 1} + 2\xi^n_{1k}\Psi^n_\alpha\Gamma^n_{\alpha 3} + \xi^n_{2k}\Psi^n_\alpha\Gamma^n_{\alpha 5}\right) K^n_k U^n_{0k} \qquad \text{(C.48)} \\
T^n_{2k} &= \left(\Gamma^n_{20} + 2\xi^n_{2k}\Gamma^n_{40} + \xi^n_{1k}\Gamma^n_{50}\right)\left(M_{kn}K^n_k - \Psi^n_\alpha U^n_{\alpha k}K^n_k\right) \\
&\quad -\left(\Psi^n_\alpha\Gamma^n_{\alpha 2} + 2\xi^n_{2k}\Psi^n_\alpha\Gamma^n_{\alpha 4} + \xi^n_{1k}\Psi^n_\alpha\Gamma^n_{\alpha 5}\right) K^n_k U^n_{0k}, \qquad \text{(C.49)}
\end{aligned}$$

where again we used our previous definition $U^n_{\nu k} = \Gamma^n_{\mu\nu}\xi^n_{\mu k}$ and the elements Q^n_k have the same form as in the local linear case.

C.2. Gradient with respect to the scale

In this section, we calculate the gradient of the reconstruction error with respect to the scale of the latent variables, that is, we calculate $\nabla_{\mathbf{s}} R(\mathrm{diag}(\mathbf{s})\mathbf{X})$. We define the elements ξ of the *scaled* design matrix as (e.g. quadratic case)

$$\xi^j_{\mu i} = \begin{cases} 1 & \mu = 0 \\ s_\mu(x_{\mu i} - x_{\mu j}) & \mu = 1, 2 \ldots q \\ s^2_{(\mu-q)}(x_{(\mu-q)i} - x_{(\mu-q)j})^2 & \mu = q+1, q+2 \ldots 2q. \end{cases} \tag{C.50}$$

Moreover, we again use the elements P^j_i and further define elements Δ^j_{mi} as in

$$\frac{\partial K^j_i}{\partial s_m} = \underbrace{2K'\left(\sum_{l=1}^{q} s^2_l (x_{li} - x_{lj})^2\right)}_{-P^j_i} \underbrace{s_m(x_{mi} - x_{mj})^2}_{\Delta^j_{mi}} = -P^j_i \Delta^j_{mi}. \tag{C.51}$$

Since we already know that the gradients with respect to $\mathbf{X}$ share many common terms across the special cases, this time we take an alternative route for the calculations: we first determine the gradient in the local constant case and then only add the respective contributions for the higher degrees.

Local constant case, independent part

The result we get here will also be part of the gradient for the higher degree cases. Therefore, we will carry the elements $\xi^j_{\nu i}$ with us, but for now we drop all derivatives of these elements, since in the local constant case $\nu = 0$ and $\xi^j_{\nu i} = 1$.

$$\begin{aligned}
\frac{\partial b^{(0)}_{ij}}{\partial s_m} &= \frac{\partial K^j_i}{\partial s_m}\xi^j_{\nu i}\Gamma^j_{\nu 0} - V^j_{i\alpha}\Gamma^j_{\alpha\mu}\sum_k \xi^j_{\mu k}\frac{\partial K^j_k}{\partial s_m}\xi^j_{\nu k}\Gamma^j_{\nu 0} \\
&= -P^j_i\Delta^j_{mi}U^j_{0i} + V^j_{i\alpha}\sum_k U^j_{\alpha k}P^j_k\Delta^j_{mk}U^j_{0k} \qquad \text{(C.52)} \\
\frac{N}{2}\frac{\partial R^{(0)}}{\partial s_m} &= \sum_{i,j} M_{ij}\frac{\partial b^{(0)}_{ij}}{\partial s_m} \\
&= -\sum_{i,j}\Delta^j_{mi}M_{ij}P^j_iU^j_{0i} + \sum_{j,k}\Delta^j_{mk}\Psi^j_\alpha U^j_{\alpha k}P^j_kU^j_{0k} \\
&= -\sum_{i,j}\Delta^j_{mi}(M_{ij} - \Psi^j_\alpha U^j_{\alpha i})P^j_iU^j_{0i} \\
&= -\sum_{i,j}\Delta^j_{mi}Q^j_i \qquad \text{(C.53)}
\end{aligned}$$

In order to write this gradient using vectors and matrices, we identify $Q^j_i = q_{ij}$ in accordance to Sec. C.1.1 and insert the definition of Δ^j_{mi} into (C.53):

$$\begin{aligned}
\frac{\partial R^{(0)}}{\partial s_m} &= -\frac{2}{N}\sum_{i,j} s_m(x_{mi} - x_{mj})^2 q_{ij} \qquad \text{(C.54)} \\
&= -\frac{2}{N}\sum_{i,j} s_m(x^2_{mi} - 2x_{mi}x_{mj} + x^2_{mj})q_{ij} \\
&= -\frac{2}{N}s_m\sum_j\left[((\mathbf{X}*\mathbf{X})\mathbf{Q})_{mj} - 2\left((\mathbf{X}\mathbf{Q})*\mathbf{X}\right)_{mj} + (\mathbf{X}*\mathbf{X})_{mj}(\mathbf{1}^T\mathbf{Q})_j\right].
\end{aligned}$$

If we now replace the summation over j by a multiplication with the vector $\mathbf{1}$ of N ones, we get the form stated in Sec. 4.4.2:

$$\nabla_{\mathbf{s}}R^{(0)} = -\frac{2}{N}\mathbf{s}*\left[(\mathbf{X}*\mathbf{X})\left(\mathbf{Q} + \operatorname{diag}(\mathbf{1}^T\mathbf{Q})\right)\mathbf{1} - 2\left((\mathbf{X}\mathbf{Q})*\mathbf{X}\right)\mathbf{1}\right]. \qquad \text{(C.55)}$$

Contribution of the linear case

Here, we only calculate the term we have to add to the gradient (C.53) from the last section, which we indicate by an upper index $^{(1-0)}$. In particular, we drop the derivatives of $K(\cdot)$ and concentrate on the dependence on ξ. We have

$$\xi^j_{\mu i} = \begin{cases} 1 & \mu = 0 \\ s_\mu(x_{\mu i} - x_{\mu j}) & \mu = 1, 2 \dots q \end{cases} \quad \text{and} \quad \frac{\partial \xi^j_{\mu i}}{\partial s_m} = \delta_{m\mu}(x_{mi} - x_{mj}). \qquad \text{(C.56)}$$

$$\begin{aligned}\frac{\partial b_{ij}^{(1-0)}}{\partial s_m} &= K_i^j \frac{\partial \xi_{\nu i}^j}{\partial s_m}\Gamma_{\nu 0}^j - V_{i\alpha}^j \Gamma_{\alpha\mu}^j \sum_k \left(\frac{\partial \xi_{\mu k}^j}{\partial s_m} K_k^j \xi_{\nu k}^j + \xi_{\mu k}^j K_k^j \frac{\partial \xi_{\nu k}^j}{\partial s_m} \right) \Gamma_{\nu 0}^j \\ &= K_i^j (x_{mi} - x_{mj}) \Gamma_{m0}^j \\ &\quad - \sum_k V_{i\alpha}^j (x_{mk} - x_{mj}) \left(\Gamma_{\alpha m}^j K_k^j U_{0k}^j + U_{\alpha k}^j K_k^j \Gamma_{m0}^j \right) \end{aligned} \tag{C.57}$$

$$\begin{aligned}\frac{N}{2}\frac{\partial R^{(1-0)}}{\partial s_m} &= \sum_{i,j} M_{ij} \frac{\partial b_{ij}^{(1-0)}}{\partial s_m} \\ &= \sum_{i,j} M_{ij} K_i^j (x_{mi} - x_{mj}) \Gamma_{m0}^j \\ &\quad - \sum_{j,k} \Psi_\alpha^j (x_{mk} - x_{mj}) \left(\Gamma_{\alpha m}^j K_k^j U_{0k}^j + U_{\alpha k}^j K_k^j \Gamma_{m0}^j \right) \\ &= \sum_{i,j} (x_{mi} - x_{mj}) \left((M_{ij} - \Psi_\alpha^j U_{\alpha i}^j) K_i^j \Gamma_{m0}^j - \Psi_\alpha^j \Gamma_{\alpha m}^j K_i^j U_{0i}^j \right) \end{aligned} \tag{C.58}$$

Please note that the term in the last brackets equals the definition (C.37) of T_{mi}^j from section C.1.2. Moreover, note that $\Delta_{mi}^j = s_m (x_{mi} - x_{mj})^2 = \xi_{mi}^j (x_{mi} - x_{mj})$, which allows us to express the complete gradient as

$$\begin{aligned}\frac{\partial R^{(1)}}{\partial s_m} &= \frac{\partial R^{(1-0)}}{\partial s_m} + \frac{\partial R^{(0)}}{\partial s_m} \\ &= \frac{2}{N} \sum_{i,j} \left(-\Delta_{mi}^j Q_i^j + (x_{mi} - x_{mj}) T_{mi}^{j(1)} \right) \\ &= \frac{2}{N} \sum_{i,j} (x_{mi} - x_{mj}) \left(-\xi_{mi}^j Q_i^j + T_{mi}^{j(1)} \right). \end{aligned} \tag{C.59}$$

Contribution of quadratic case without cross-terms

Please recall the definition (C.50) of the scaled design matrix, from which we have

$$\frac{\partial \xi_{\nu i}^j}{\partial s_m} = \delta_{m\nu} (x_{mi} - x_{mj}) + 2\delta_{(m+q)\nu} s_m (x_{mi} - x_{mj})^2 \tag{C.60}$$

We already handled the first summand in the local linear case, and therefore only take the second contribution $2\delta_{(m+q)\nu} s_m (x_{mi} - x_{mj})^2 = 2\delta_{(m+q)\nu} \Delta_{mi}^j$ into account.

$$\begin{aligned}\frac{\partial b_{ij}^{(2-1)}}{\partial s_m} &= K_i^j 2\delta_{(m+q)\nu} \Delta_{mi}^j \Gamma_{\nu 0}^j \\ &\quad - V_{i\alpha}^j \Gamma_{\alpha\mu}^j \sum_k 2\Delta_{mk}^j \left(\delta_{(m+q)\mu} K_k^j \xi_{\nu k}^j + \xi_{\mu k}^j K_k^j \delta_{(m+q)\nu} \right) \Gamma_{\nu 0}^j \\ &= 2K_i^j \Delta_{mi}^j \Gamma_{(m+q)0}^j \\ &\quad - 2V_{i\alpha}^j \sum_k \Delta_{mk}^j \left(\Gamma_{\alpha(m+q)}^j K_k^j U_{0k}^j + U_{\alpha k}^j K_k^j \Gamma_{(m+q)0}^j \right) \end{aligned} \tag{C.61}$$

$$\begin{aligned}
\frac{N}{2}\frac{\partial R^{(2-1)}}{\partial s_m} &= \sum_{i,j} 2\Delta^j_{mi}\left((M_{ij} - \Psi^j_\alpha U^j_{\alpha i})K^j_i\Gamma^j_{(m+q)0} - \Psi^j_\alpha \Gamma^j_{\alpha(m+q)}K^j_i U^j_{0i}\right) \\
&= \sum_{i,j}(x_{mi} - x_{mj})\Big((M_{ij} - \Psi^j_\alpha U^j_{\alpha i})K^j_i 2\xi^j_{mi}\Gamma^j_{(m+q)0} \\
&\qquad\qquad -\Psi^j_\alpha 2\xi^j_{mi}\Gamma^j_{\alpha(m+q)}K^j_i U^j_{0i}\Big) \qquad \text{(C.62)} \\
&= \sum_{i,j}(x_{mi} - x_{mj})(T^{j(2)}_{mi} - T^{j(1)}_{mi}). \qquad \text{(C.63)}
\end{aligned}$$

Here, $T^{j(2)}_{mi}$ stands for (C.43) from Sec. C.1.3, and $T^{j(1)}_{mi}$ is (C.37) from the local linear case (Sec. C.1.2). In analogy to the preceding section, the complete gradient can be expressed as

$$\begin{aligned}
\frac{\partial R^{(2)}}{\partial s_m} &= \frac{\partial R^{(2-1)}}{\partial s_m} + \frac{\partial R^{(1-0)}}{\partial s_m} + \frac{\partial R^{(0)}}{\partial s_m} \\
&= \frac{2}{N}\sum_{i,j}\left(-\Delta^j_{mi}Q^j_i + (x_{mi} - x_{mj})T^{j(2)}_{mi}\right) \\
&= \frac{2}{N}\sum_{i,j}(x_{mi} - x_{mj})\left(-\xi^j_{mi}Q^j_i + T^{j(2)}_{mi}\right). \qquad \text{(C.64)}
\end{aligned}$$

Contribution of the full quadratic case (q=2)

Since the calculation is very similar to the case without the cross-term, we only present the resulting gradient:

$$\frac{\partial R^{(22)}}{\partial s_m} = \frac{2}{N}\sum_{i,j}(x_{mi} - x_{mj})\left(-\xi^j_{mi}Q^j_i + T^{j(22)}_{mi}\right), \qquad \text{(C.65)}$$

where $T^{j(22)}_{mi}$ is given by (C.48) and (C.49), depending on $m = 1, 2$.

C.3. Gradient calculation for the landmark variant

Even while no corresponding experiments have been presented in this thesis, this section contains the calculation of the gradient for landmark ULPR, that is, a landmark variant of UKR with a general polynomial degree. The gradient of landmark UKR as presented in Sec. 5.3.1 is the special case of local constant regression.

Again, we write the regression function in a way similar to the standard UKR function:

$$\mathbf{f}(\mathbf{x}; \hat{\mathbf{X}}) = \mathbf{m}_\mathbf{x} = \hat{\mathbf{Y}}\mathbf{W}\bar{\mathbf{X}}^T\mathbf{C}^{-1}\mathbf{e}_0 = \hat{\mathbf{Y}}\mathbf{b}(\mathbf{x}; \hat{\mathbf{X}}), \qquad \text{(C.66)}$$

which allows us to write the reconstruction error in the form

$$R(\mathbf{X}) = \frac{1}{N}\sum_{j=1}^{N}\|\mathbf{y}_j - \mathbf{f}(\mathbf{x}_j; \hat{\mathbf{X}})\|^2 = \frac{1}{N}\|\mathbf{Y} - \hat{\mathbf{Y}}\mathbf{B}(\mathbf{X}, \hat{\mathbf{X}})\|^2_F. \qquad \text{(C.67)}$$

Now we need expressions for the columns $\mathbf{b}(\mathbf{x}_j; \hat{\mathbf{X}})$ of $\mathbf{B}(\mathbf{X}, \hat{\mathbf{X}})$ and its derivatives, for which we introduce the same shortcut notation as in Sec. C.1:

$$\mathbf{b}(\mathbf{x}_j; \hat{\mathbf{X}}) = \mathbf{W}^j \bar{\mathbf{X}}^{j^T} \mathbf{C}^{j^{-1}} \mathbf{e}_0 = \mathbf{V}^j \mathbf{C}^{j^{-1}} \mathbf{e}_0. \tag{C.68}$$

An upper index j denotes the dependence of a matrix or vector on the position $\mathbf{x}_j$ where we evaluate the regression function. $\mathbf{W}^j$ is a diagonal matrix with the entries

$$(\mathbf{W}^j)_{il} = \delta_{il} K(\hat{\mathbf{x}}_i - \mathbf{x}_j) = \delta_{il} K_i^j. \tag{C.69}$$

For the entries of the matrix $\bar{\mathbf{X}}^j$, we introduce the notation $(\bar{\mathbf{X}}^j)_{\mu i} = \xi_{\mu i}^j$. In the local linear case, we have

$$\xi_{\mu i}^j = \begin{pmatrix} 1 \\ \hat{\mathbf{x}}_i - \mathbf{x}_j \end{pmatrix}_\mu$$

and μ ranges from 0 to q, whereas in the quadratic case ($q = 1$), we have

$$\xi_{\mu i}^j = \begin{pmatrix} 1 \\ \hat{x}_i - x_j \\ (\hat{x}_i - x_j)^2 \end{pmatrix}_\mu$$

and μ ranges from 0 to 2. For the local constant case (Nadaraya-Watson), we simply have $\xi_i^j = 1$. In the following, the number of data vectors and landmark points are denoted by N and $\hat{N}$, respectively. If not noted otherwise, summations over latin indices are understood to range from 1 to $\hat{N}$. We again use the convention to automatically sum over double greek indices.

$$\begin{aligned} \frac{\partial R(\mathbf{X})}{\partial x_{mn}} &= \frac{2}{N} \sum_{i=1}^{\hat{N}} \sum_{j=1}^{N} \left(\hat{\mathbf{Y}}^T (\mathbf{Y} - \hat{\mathbf{Y}} \mathbf{B}(\mathbf{X}, \hat{\mathbf{X}}) \right)_{ij} \frac{\partial b_{ij}(\mathbf{X}, \hat{\mathbf{X}})}{\partial x_{mn}} \\ &= \frac{2}{N} \sum_{i} \sum_{j=1}^{N} M_{ij} \frac{\partial b_{ij}}{\partial x_{mn}} \end{aligned} \tag{C.70}$$

$$\frac{\partial R(\mathbf{X})}{\partial \hat{x}_{mn}} = \frac{2}{N} \sum_{i} \sum_{j=1}^{N} M_{ij} \frac{\partial b_{ij}}{\partial \hat{x}_{mn}} \tag{C.71}$$

$$\frac{\partial b_{ij}}{\partial x_{mn}} = \left[\frac{\partial V_{i\nu}^j}{\partial x_{mn}} - V_{i\alpha}^j \Gamma_{\alpha\mu}^j \frac{\partial C_{\mu\nu}^j}{\partial x_{mn}} \right] \Gamma_{\nu 0}^j \tag{C.72}$$

$$\frac{\partial b_{ij}}{\partial \hat{x}_{mn}} = \left[\frac{\partial V_{i\nu}^j}{\partial \hat{x}_{mn}} - V_{i\alpha}^j \Gamma_{\alpha\mu}^j \frac{\partial C_{\mu\nu}^j}{\partial \hat{x}_{mn}} \right] \Gamma_{\nu 0}^j \tag{C.73}$$

$$\begin{aligned} \frac{\partial K_i^j}{\partial x_{mn}} &= 2K'(\|\hat{\mathbf{x}}_i - \mathbf{x}_j\|^2)(\hat{x}_{mi} - x_{mj})(-\delta_{jn}) \\ &= -P_i^j (\hat{x}_{mi} - x_{mj})(-\delta_{jn}) \end{aligned} \tag{C.74}$$

$$\begin{aligned} \frac{\partial K_i^j}{\partial \hat{x}_{mn}} &= 2K'(\|\hat{\mathbf{x}}_i - \mathbf{x}_j\|^2)(\hat{x}_{mi} - x_{mj})\delta_{in} \\ &= -P_i^j (\hat{x}_{mi} - x_{mj})\delta_{in} \end{aligned} \tag{C.75}$$

A general form of the derivatives of the variables $\xi^j_{\nu i}$ can be given as

$$\frac{\partial \xi^j_{\nu i}}{\partial x_{mn}} = \chi^{ij}_{m\nu}(-\delta_{jn}) \quad \text{and} \quad \frac{\partial \xi^j_{\nu i}}{\partial \hat{x}_{mn}} = \chi^{ij}_{m\nu}\delta_{in}. \tag{C.76}$$

$$\begin{aligned}
\frac{\partial V^j_{i\nu}}{\partial x_{mn}} &= \frac{\partial (K^j_i \xi^j_{\nu i})}{\partial x_{mn}} = \frac{\partial K^j_i}{\partial x_{mn}}\xi^j_{\nu i} + K^j_i \frac{\partial \xi^j_{\nu i}}{\partial x_{mn}} \\
&= \left[-P^j_i(\hat{x}_{mi} - x_{mj})\xi^j_{\nu i} + K^j_i \chi^{ij}_{m\nu}\right](-\delta_{jn}) \qquad \text{(C.77)} \\
\frac{\partial V^j_{i\nu}}{\partial \hat{x}_{mn}} &= \left[-P^j_i(\hat{x}_{mi} - x_{mj})\xi^j_{\nu i} + K^j_i \chi^{ij}_{m\nu}\right]\delta_{in} \qquad \text{(C.78)}
\end{aligned}$$

$$\begin{aligned}
\frac{\partial C^j_{\mu\nu}}{\partial x_{mn}} &= \sum_i \left[\frac{\partial \xi^j_{\mu i}}{\partial x_{mn}} K^j_i \xi^j_{\nu i} + \xi^j_{\mu i}\frac{\partial K^j_i}{\partial x_{mn}}\xi^j_{\nu i} + \xi^j_{\mu i} K^j_i \frac{\partial \xi^j_{\nu i}}{\partial x_{mn}}\right] \\
&= \sum_i \left[\chi^{ij}_{m\mu} K^j_i \xi^j_{\nu i} - \xi^j_{\mu i} P^j_i(\hat{x}_{mi} - x_{mj})\xi^j_{\nu i} + \xi^j_{\mu i} K^j_i \chi^{ij}_{m\nu}\right](-\delta_{jn}) \\
&= -\delta_{jn}\sum_k \left[\Lambda^{kj}_{m\mu\nu} - \xi^j_{\mu k} P^j_k(\hat{x}_{mk} - x_{mj})\xi^j_{\nu k}\right] \qquad \text{(C.79)} \\
\frac{\partial C^j_{\mu\nu}}{\partial \hat{x}_{mn}} &= \sum_i \left[\chi^{ij}_{m\mu} K^j_i \xi^j_{\nu i} - \xi^j_{\mu i} P^j_i(\hat{x}_{mi} - x_{mj})\xi^j_{\nu i} + \xi^j_{\mu i} K^j_i \chi^{ij}_{m\nu}\right]\delta_{in} \\
&= \Lambda^{nj}_{m\mu\nu} - \xi^j_{\mu n} P^j_n(\hat{x}_{mn} - x_{mj})\xi^j_{\nu n} \qquad \text{(C.80)}
\end{aligned}$$

Here, we again used the definition

$$\Lambda^{ij}_{m\mu\nu} = \chi^{ij}_{m\mu} K^j_i \xi^j_{\nu i} + \xi^j_{\mu i} K^j_i \chi^{ij}_{m\nu} = K^j_i(\chi^{ij}_{m\mu}\xi^j_{\nu i} + \chi^{ij}_{m\nu}\xi^j_{\mu i}). \tag{C.81}$$

Inserting the derivatives of C and V into (C.72) and (C.73) yields

$$\begin{aligned}
\frac{\partial b_{ij}}{\partial x_{mn}} &= \left[-P^j_i(\hat{x}_{mi} - x_{mj})\xi^j_{\nu i}\Gamma^j_{\nu 0} + K^j_i \chi^{ij}_{m\nu}\Gamma^j_{\nu 0}\right](-\delta_{jn}) \\
&\quad + \delta_{jn}\sum_k \left[V^j_{i\alpha}\Gamma^j_{\alpha\mu}\Lambda^{kj}_{m\mu\nu}\Gamma^j_{\nu 0} - V^j_{i\alpha}\Gamma^j_{\alpha\mu}\xi^j_{\mu k}P^j_k(\hat{x}_{mk} - x_{mj})\xi^j_{\nu k}\Gamma^j_{\nu 0}\right] \qquad \text{(C.82)}
\end{aligned}$$

and

$$\begin{aligned}
\frac{\partial b_{ij}}{\partial \hat{x}_{mn}} &= \left[-P^j_i(\hat{x}_{mi} - x_{mj})\xi^j_{\nu i}\Gamma^j_{\nu 0} + K^j_i \chi^{ij}_{m\nu}\Gamma^j_{\nu 0}\right]\delta_{in} \\
&\quad - V^j_{i\alpha}\Gamma^j_{\alpha\mu}\Lambda^{nj}_{m\mu\nu}\Gamma^j_{\nu 0} + V^j_{i\alpha}\Gamma^j_{\alpha\mu}\xi^j_{\mu n}P^j_n(\hat{x}_{mn} - x_{mj})\xi^j_{\nu n}\Gamma^j_{\nu 0}, \qquad \text{(C.83)}
\end{aligned}$$

which together nicely sum up to the derivatives (C.17) of b_{ij} in the non-landmark case.

We now insert (C.82) into (C.70):

$$\begin{aligned}
\frac{N}{2}\frac{\partial R(\mathbf{X})}{\partial x_{mn}} &= \sum_i \sum_{j=1}^{N} M_{ij}\left[-P_i^j(\hat{x}_{mi}-x_{mj})\xi_{\nu i}^j\Gamma_{\nu 0}^j + K_i^j\chi_{m\nu}^{ij}\Gamma_{\nu 0}^j\right](-\delta_{jn}) \\
&\quad + \sum_i \sum_{j=1}^{N} M_{ij}\delta_{jn}\sum_k V_{i\alpha}^j\Gamma_{\alpha\mu}^j\Lambda_{m\mu\nu}^{kj}\Gamma_{\nu 0}^j \\
&\quad - \sum_i \sum_{j=1}^{N} M_{ij}\delta_{jn}\sum_k V_{i\alpha}^j\Gamma_{\alpha\mu}^j\xi_{\mu k}^j P_k^j(\hat{x}_{mk}-x_{mj})\xi_{\nu k}^j\Gamma_{\nu 0}^j \\
&= \sum_i M_{in}\left[P_i^n(\hat{x}_{mi}-x_{mn})\xi_{\nu i}^n\Gamma_{\nu 0}^n - K_i^n\chi_{m\nu}^{in}\Gamma_{\nu 0}^n\right] \\
&\quad + \sum_{i,k} M_{in}V_{i\alpha}^n\Gamma_{\alpha\mu}^n\Lambda_{m\mu\nu}^{kn}\Gamma_{\nu 0}^n - \sum_{i,k} M_{in}V_{i\alpha}^n\Gamma_{\alpha\mu}^n\xi_{\mu k}^n P_k^n(\hat{x}_{mk}-x_{mn})\xi_{\nu k}^n\Gamma_{\nu 0}^n \\
&= \sum_k M_{kn}\left[P_k^n(\hat{x}_{mk}-x_{mn})\xi_{\nu k}^n\Gamma_{\nu 0}^n - K_k^n\chi_{m\nu}^{kn}\Gamma_{\nu 0}^n\right] \\
&\quad + \sum_k \Psi_\alpha^n\Gamma_{\alpha\mu}^n\Lambda_{m\mu\nu}^{kn}\Gamma_{\nu 0}^n - \sum_k \Psi_\alpha^n\Gamma_{\alpha\mu}^n\xi_{\mu k}^n P_k^n(\hat{x}_{mk}-x_{mn})\xi_{\nu k}^n\Gamma_{\nu 0}^n \\
&= \sum_k (\hat{x}_{mk}-x_{mn})\left[M_{kn}P_k^n\xi_{\nu k}^n\Gamma_{\nu 0}^n - \Psi_\alpha^n\Gamma_{\alpha\mu}^n\xi_{\mu k}^n P_k^n\xi_{\nu k}^n\Gamma_{\nu 0}^n\right] \\
&\quad - \sum_k M_{kn}K_k^n\chi_{m\nu}^{kn}\Gamma_{\nu 0}^n + \sum_k \Psi_\alpha^n\Gamma_{\alpha\mu}^n\Lambda_{m\mu\nu}^{kn}\Gamma_{\nu 0}^n \\
&= \sum_k (\hat{x}_{mk}-x_{mn})Q_k^n - \sum_k T_{mk}^n. \qquad \text{(C.84)}
\end{aligned}$$

A very similar expression results from inserting (C.83) into (C.71):

$$\begin{aligned}
\frac{N}{2}\frac{\partial R(\hat{\mathbf{X}})}{\partial \hat{x}_{mn}} &= \sum_{i=1}^{\hat{N}} \sum_{j=1}^{N} M_{ij}\Big\{\left[-P_i^j(\hat{x}_{mi}-x_{mj})\xi_{\nu i}^j\Gamma_{\nu 0}^j + K_i^j\chi_{m\nu}^{ij}\Gamma_{\nu 0}^j\right]\delta_{in} \\
&\quad - V_{i\alpha}^j\Gamma_{\alpha\mu}^j\Lambda_{m\mu\nu}^{nj}\Gamma_{\nu 0}^j + V_{i\alpha}^j\Gamma_{\alpha\mu}^j\xi_{\mu n}^j P_n^j(\hat{x}_{mn}-x_{mj})\xi_{\nu n}^j\Gamma_{\nu 0}^j\Big\} \\
&= \sum_j M_{nj}\left[-P_n^j(\hat{x}_{mn}-x_{mj})\xi_{\nu n}^j\Gamma_{\nu 0}^j + K_n^j\chi_{m\nu}^{nj}\Gamma_{\nu 0}^j\right] \\
&\quad - \sum_j \Psi_\alpha^j\Gamma_{\alpha\mu}^j\Lambda_{m\mu\nu}^{nj}\Gamma_{\nu 0}^j + \sum_j \Psi_\alpha^j\Gamma_{\alpha\mu}^j\xi_{\mu n}^j P_n^j(\hat{x}_{mn}-x_{mj})\xi_{\nu n}^j\Gamma_{\nu 0}^j \\
&= \sum_j (\hat{x}_{mn}-x_{mj})\left[-M_{nj}P_n^j\xi_{\nu n}^j\Gamma_{\nu 0}^j + \Psi_\alpha^j\Gamma_{\alpha\mu}^j\xi_{\mu n}^j P_n^j\xi_{\nu n}^j\Gamma_{\nu 0}^j\right] \\
&\quad + \sum_j M_{nj}K_n^j\chi_{m\nu}^{nj}\Gamma_{\nu 0}^j - \sum_j \Psi_\alpha^j\Gamma_{\alpha\mu}^j\Lambda_{m\mu\nu}^{nj}\Gamma_{\nu 0}^j \qquad \text{(C.85)} \\
&= \sum_j (\hat{x}_{mn}-x_{mj})(-Q_n^j) + \sum_j T_{mn}^j. \qquad \text{(C.86)}
\end{aligned}$$

In both expression, we used the same abbreviations Q_n^j and T_{mn}^j as already defined in

the non-landmark variant (Sec. C.1). The special cases of local constant, linear, and quadratic regression can be calculated in perfect analogy.

In particular, in the local constant case we have $T^j_{mn} = 0$ and $Q^j_n = q_{nj}$ are the elements of the matrix (cf. Sec. 5.3.1)

$$\mathbf{Q} = \mathbf{P} * \left[\mathbf{M} - \mathbf{1}_n \mathbf{1}_N^T (\mathbf{B} * \mathbf{M})\right]. \tag{C.87}$$

We therefore can write

$$\frac{\partial R(\mathbf{X}, \hat{\mathbf{X}})}{\partial x_{mn}} = \frac{2}{N} \sum_{k=1}^{\hat{N}} (\hat{x}_{mk} - x_{mn}) q_{kn} = \frac{2}{N} \left(\hat{\mathbf{X}}\mathbf{Q} - \mathbf{X} \cdot \mathrm{diag}(\mathbf{1}_n^T \mathbf{Q})\right)_{mn} \tag{C.88}$$

$$\frac{\partial R(\mathbf{X}, \hat{\mathbf{X}})}{\partial \hat{x}_{mn}} = \frac{2}{N} \sum_{j=1}^{N} (\hat{x}_{mn} - x_{mj})(-q_{jn}) = \frac{2}{N} \left(\mathbf{X}\mathbf{Q} - \hat{\mathbf{X}} \cdot \mathrm{diag}(\mathbf{1}_N^T \mathbf{Q}^T)\right)_{mn} \tag{C.89}$$

in accordance to the matrix expressions given in Sec. 5.3.1.

C.4. Jacobi matrix for projecting new data

For projecting new data onto an UKR or ULPR manifold, it is advantageous to apply a least-squares method like the Levenberg-Marquardt algorithm. To this aim, we need expressions for the Jacobian $\nabla_{\mathbf{x}} \mathbf{f}(\mathbf{x}; \mathbf{X}) = \mathbf{Y} \nabla_{\mathbf{x}} \mathbf{b}(\mathbf{x}; \mathbf{X})$, but luckily we can "recycle" our calculations for the landmark variant in the way of

$$J_{im} = \frac{\partial b_i(\mathbf{x}, \hat{\mathbf{X}})}{\partial x_m} = \frac{\partial b_i(\mathbf{X}, \hat{\mathbf{X}})}{\partial x_{m1}}. \tag{C.90}$$

We therefore take the results from (C.72) where we just set $n = 1$ and drop the corresponding upper index.

$$\begin{aligned}
J_{im} &= P_i(\hat{x}_{mi} - x_m)\xi_{\nu i}\Gamma_{\nu 0} - K_i \chi^{i1}_{m\nu}\Gamma_{\nu 0} \\
&\quad + \sum_k \left[V_{i\alpha}\Gamma_{\alpha\mu}\Lambda^{k1}_{m\mu\nu}\Gamma_{\nu 0} - V_{i\alpha}\Gamma_{\alpha\mu}\xi_{\mu k} P_k(\hat{x}_{mk} - x_m)\xi_{\nu k}\Gamma_{\nu 0}\right] && \text{(C.91)} \\
&= P_i(\hat{x}_{mi} - x_m)\xi_{\nu i}\Gamma_{\nu 0} - K_i \chi^{i1}_{m\nu}\Gamma_{\nu 0} \\
&\quad + \sum_k \left[K_i\xi_{\alpha i}\Gamma_{\alpha\mu}\Lambda^{k1}_{m\mu\nu}\Gamma_{\nu 0} - K_i\xi_{\alpha i}\Gamma_{\alpha\mu}\xi_{\mu k} P_k(\hat{x}_{mk} - x_m)\xi_{\nu k}\Gamma_{\nu 0}\right] && \text{(C.92)} \\
&= P_i(\hat{x}_{mi} - x_m)U_{0i} - K_i \chi^{i1}_{m\nu}\Gamma_{\nu 0} \\
&\quad + \sum_k \left[K_i U_{\mu i}\Lambda^{k1}_{m\mu\nu}\Gamma_{\nu 0} - K_i U_{\mu i}\xi_{\mu k} P_k(\hat{x}_{mk} - x_m)U_{0k}\right] && \text{(C.93)} \\
&= G_{im} - K_i \chi^{i1}_{m\nu}\Gamma_{\nu 0} + \sum_k H_{i\mu}\Lambda^{k1}_{m\mu\nu}\Gamma_{\nu 0} - \sum_k H_{i\mu}\xi_{\mu k} G_{km} && \text{(C.94)}
\end{aligned}$$

Here we used the abbreviations

$$G_{im} = P_i(\hat{x}_{mi} - x_m)U_{0i} \quad \text{and} \quad H_{i\mu} = K_i U_{\mu i} = K_i \xi_{\nu i}\Gamma_{\nu\mu}. \tag{C.95}$$

C.4.1. Local constant estimate

$$\xi = 1 \quad , \quad \Lambda = \chi = 0 \quad , \quad U_{0*} = \Gamma_{00} = \frac{1}{\sum_i K_i} \tag{C.96}$$

$$J_{im} = \frac{P_i}{\sum_j K_j}(\hat{x}_{mi} - x_m) - \frac{K_i}{\sum_j K_j}\sum_k \frac{P_k}{\sum_j K_j}(\hat{x}_{mk} - x_m) \tag{C.97}$$

C.4.2. Local linear estimate

$$\chi^{ij}_{m\mu} = \delta_{m\mu} \text{ and } \Lambda^{ij}_{m\mu\nu} = K^j_i(\delta_{m\mu}\xi^j_{\nu i} + \delta_{m\nu}\xi^j_{\mu i}). \tag{C.98}$$

$$\begin{aligned} J_{im} &= G_{im} - K_i\delta_{m\nu}\Gamma_{\nu 0} \\ &\quad + \sum_k H_{i\mu}K_k(\delta_{m\mu}\xi_{\nu k} + \delta_{m\nu}\xi_{\mu k})\Gamma_{\nu 0} - \sum_k H_{i\mu}\xi_{\mu k}G_{km} \\ &= G_{im} - K_i\Gamma_{m0} \\ &\quad + \sum_k H_{im}K_k\xi_{\nu k}\Gamma_{\nu 0} + \sum_k H_{i\mu}K_k\xi_{\mu k}\Gamma_{m0} - \sum_k H_{i\mu}\xi_{\mu k}G_{km} \\ &= G_{im} - K_i\Gamma_{m0} + H_{im}\sum_k H_{k0} + H_{i\mu}\sum_k \xi_{\mu k}\left(K_k\Gamma_{m0} - G_{km}\right) \end{aligned} \tag{C.99}$$

C.4.3. Local quadratic estimate w/o cross-terms

We have

$$\chi^i_{m\mu} = \delta_{m\mu} + 2\xi_{mi}\delta_{m(\mu-q)} \tag{C.100}$$

and thus

$$\Lambda^k_{m\mu\nu} = K_k\left[\left(\delta_{m\mu} + 2\xi_{mk}\delta_{m(\mu-q)}\right)\xi_{\nu k} + \left(\delta_{m\nu} + 2\xi_{mk}\delta_{m(\nu-q)}\right)\xi_{\mu k}\right]. \tag{C.101}$$

$$\begin{aligned} J_{im} &= G_{im} - K_i(\delta_{m\nu} + 2\xi_{mi}\delta_{m(\nu-q)})\Gamma_{\nu 0} \\ &\quad + \sum_k H_{i\mu}K_k\left(\delta_{m\mu} + 2\xi_{mk}\delta_{m(\mu-q)}\right)\xi_{\nu k}\Gamma_{\nu 0} \\ &\quad + \sum_k H_{i\mu}K_k\left(\delta_{m\nu} + 2\xi_{mk}\delta_{m(\nu-q)}\right)\xi_{\mu k}\Gamma_{\nu 0} - \sum_k H_{i\mu}\xi_{\mu k}G_{km} \\ &= G_{im} - K_i\Gamma_{m0} - 2K_i\xi_{mi}\Gamma_{(m+q)0} \\ &\quad + H_{im}\sum_k K_kU_{0k} + 2H_{i(m+q)}\underbrace{\sum_k K_k\xi_{mk}U_{0k}}_{=C_{m\nu}\Gamma_{\nu 0}=_{m0}=0} \\ &\quad + H_{i\mu}\sum_k \xi_{\mu k}K_k\left(\Gamma_{m0} + 2\xi_{mk}\Gamma_{(m+q)0}\right) - \sum_k H_{i\mu}\xi_{\mu k}G_{km} \\ &= G_{im} - K_i\Gamma_{m0} - 2K_i\xi_{mi}\Gamma_{(m+q)0} + H_{im}\sum_k H_{k0} \\ &\quad + H_{i\mu}\sum_k \xi_{\mu k}\left(K_k\Gamma_{m0} + 2K_k\xi_{mk}\Gamma_{(m+q)0} - G_{km}\right) \end{aligned} \tag{C.102}$$

C.4.4. Local quadratic estimate with cross-term, q=2

Here we have

$$\chi^i_{1\mu} = \delta_{1\mu} + 2\xi_{1i}\delta_{3\mu} + 2\xi_{2i}\delta_{5\mu} \tag{C.103}$$

$$\chi^i_{2\mu} = \delta_{2\mu} + 2\xi_{2i}\delta_{4\mu} + 2\xi_{1i}\delta_{5\mu} \tag{C.104}$$

and thus

$$\Lambda^k_{1\mu\nu} = K_k \left[\left(\delta_{1\mu} + 2\xi_{1k}\delta_{3\mu} + \xi_{2k}\delta_{5\mu}\right)\xi_{\nu k} + \left(\delta_{1\nu} + 2\xi_{1k}\delta_{3\nu} + \xi_{2k}\delta_{5\nu}\right)\xi_{\mu k} \right]. \tag{C.105}$$

$$J_{i1} = G_{i1} - K_i\chi^{i1}_{1\nu}\Gamma_{\nu 0} + \sum_k H_{i\mu}\Lambda^{k1}_{1\mu\nu}\Gamma_{\nu 0} - \sum_k H_{i\mu}\xi_{\mu k}G_{k1} \tag{C.106}$$

$$\begin{aligned} &= G_{i1} - K_i(\Gamma_{10} + 2\xi_{1i}\Gamma_{30} + 2\xi_{2i}\Gamma_{50}) \\ &\quad + H_{i1}\underbrace{\sum_k K_k\xi_{\nu k}\Gamma_{\nu 0}}_{=H_{k0}} + 2H_{i3}\underbrace{\sum_k K_k\xi_{1k}\xi_{\nu k}\Gamma_{\nu 0}}_{=C_{1\nu}\Gamma_{\nu 0}=0} + 2H_{i5}\underbrace{\sum_k K_k\xi_{2k}\xi_{\nu k}\Gamma_{\nu 0}}_{=C_{2\nu}\Gamma_{\nu 0}=0} \\ &\quad + H_{i\mu}\sum_k \xi_{\mu k}K_k(\Gamma_{10} + 2\xi_{1k}\Gamma_{30} + 2\xi_{2k}\Gamma_{50}) \\ &\quad - \sum_k H_{i\mu}\xi_{\mu k}G_{k1} \end{aligned} \tag{C.107}$$

$$\begin{aligned} &= G_{i1} - K_i(\Gamma_{10} + 2\xi_{1i}\Gamma_{30} + 2\xi_{2i}\Gamma_{50}) + H_{i1}\sum_k H_{k0} \\ &\quad + H_{i\mu}\sum_k \xi_{\mu k}\left[K_k(\Gamma_{10} + 2\xi_{1k}\Gamma_{30} + 2\xi_{2k}\Gamma_{50}) - G_{k1}\right] \end{aligned} \tag{C.108}$$

$$\begin{aligned} J_{i2} &= G_{i2} - K_i(\Gamma_{20} + 2\xi_{2i}\Gamma_{40} + 2\xi_{1i}\Gamma_{50}) + H_{i2}\sum_k H_{k0} \\ &\quad + H_{i\mu}\sum_k \xi_{\mu k}\left[K_k(\Gamma_{20} + 2\xi_{2k}\Gamma_{40} + 2\xi_{1k}\Gamma_{50}) - G_{k2}\right] \end{aligned} \tag{C.109}$$

References

Aizerman, M., Braverman, E., and Rozonoer, L. (1964). Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning. *Automation and Remote Control*, *25*, 821 - 837.

Allgower, E. L., and Georg, K. (2003). *Introduction to Numerical Continuation Methods.* Classics in Applied Mathematics (Vol. 45). Philadelphia: SIAM.

Allwein, E. L., Schapire, R. E., and Singer, Y. (2000). Reducing Multiclass to Binary: A Unifying Approach for Margin Classifiers. *Journal of Machine Learning Research*, *1*, 113–141.

Alzate, C., and Suykens, J. A. K. (2005). Extending Kernel Principal Component Analysis to General Underlying Loss Functions. *1*, 214-219.

Bakir, G. H., Weston, J., and Schölkopf, B. (2003). Learning to find pre-images . In *Advances in Neural Information Processing Systems* (Vol. 15). Cambridge, MA: MIT Press.

Baldi, P., and Hornik, K. (1989). Neural Networks and Principal Component Analysis: Learning from Examples without Local Minima. *Neural Networks*, *2*(1), 53–58.

Bartholomew, D. J., and Knott, M. (1999). *Latent Variable Models and Factor Analysis.* Kendall's Library of Statistics (2nd ed., Vol. 7). London: Arnold.

Bates, D. M., and Watts, D. G. (1988). *Nonlinear Regression Analysis and its Applications.* New York: Wiley.

Belkin, M., and Niyogi, P. (2002). *Laplacian Eigenmaps for Dimensionality Reduction and Data Representation* (Technical Report No. TR 2002-01). Univ. Chicago, Dept. Comp. Sci. and Statistics.

Belkin, M., and Niyogi, P. (2003). Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation*, *15 (6)*, 1373–1396.

Bellman, R. E. (1961). *Adaptive Control Processes.* Princeton, NY: Princeton University Press.

Bengio, Y., Delalleau, O., Paiement, J.-F., Vincent, P., and Ouimet, M. (2004). Learning Eigenfunctions Links Spectral Embedding and Kernel PCA. *Neural Computation*,

16, 2197–2219.

Bengio, Y., Paiement, J., Vincent, P., Delalleau, O., Roux, N. L., and Ouimet, M. (2004). Out-of-Sample Extensions for LLE, Isomap, MDS, Eigenmaps, and Spectral Clustering. In S. Thrun, L. Saul, and B. Schölkopf (Eds.), *Advances in Neural Information Processing Systems* (Vol. 16). Cambridge, MA: MIT Press.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford: Clarendon Press.

Bishop, C. M., Svensén, M., and Williams, C. K. I. (1997). GTM: A Principled Alternative to the Self-Organizing Map. In M. C. Mozer, M. I. Jordan, and T. Petsche (Eds.), *Advances in Neural Information Processing Systems* (Vol. 9, pp. 354–360). Cambridge, MA: MIT Press.

Bishop, C. M., Svensén, M., and Williams, C. K. I. (1998a). Developments of the Generative Topographic Mapping. *Neurocomputing*, *21*, 203–224.

Bishop, C. M., Svensén, M., and Williams, C. K. I. (1998b). GTM: The Generative Topographic Mapping. *Neural Computation*, *10*(1), 215–234.

Boser, B. E., Guyon, I., and Vapnik, V. (1992). A Training Algorithm for Optimal Margin Classifiers. In D. Haussler (Ed.), *5th Annual ACM Workshop on Computational Learing Theory* (pp. 144–152). Pittsburgh, PA: ACM Press.

Bourlard, H., and Kamp, Y. (1988). Auto-association by Multilayer Perceptrons and Singular Value Decomposition. *Biological Cybernetics*, *59*, 291–294.

Bregler, C., and Omohundro, S. M. (1994). Surface Learning with Applications to Lipreading. In J. D. Cowan, G. Tesauro, and J. Alspector (Eds.), *Advances in Neural Information Processing Systems* (Vol. 6, pp. 43–50). San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Breiman, L., and Spector, P. (1992). Submodel Selection and Evaluation in Regression. The X-Random Case. *International Statistical Review*, *60*(3), 291–319.

Chang, H., and Yeung, D.-Y. (2006). Robust Locally Linear Embedding. *Pattern Recognition*, *39*, 1053–1065.

Chang, K., and Ghosh, J. (1999). Probabilistic Principal Surfaces. In *Proc. IEEE Int. Joint Conf. Neural Networks.*

Chang, K., and Ghosh, J. (2001). A Unified Model for Probabilistic Principal Surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *23*(1), 22–41.

Cherkassky, V., and Mulier, F. (1998). *Learning from Data: Concepts, Theory, and*

Methods. New York: Wiley.

Clark, R. M. (1975). A Calibration Curve for Radiocarbon Dates. *Antiquity*, *49*(196), 251-266.

Cleveland, W. S. (1979). Robust Locally Weighted Regression and Smoothing Scatterplots. *Journal of the American Statistical Association*, *74*, 829–836.

Cleveland, W. S., and Loader, C. L. (1996). Smoothing by Local Regression: Principles and Methods. In W. Härdle and M. G. Schimek (Eds.), *Statistical theory and computational aspects of smoothing* (pp. 10–49). New York: Springer.

de Boor, C. (1978). *Practical Guide to Splines*. New York: Springer.

de Silva, V., and Tenenbaum, B. (2004). *Sparse Multidimensional Scaling using Landmark Points* (Technical Report). Stanford University.

Demartines, P., and Hérault, J. (1997). Curvilinear Component Analysis: A Self-Organizing Neural Network for Nonlinear Mapping of Data Sets. *IEEE Transactions on Neural Networks*, *8*(1), 148–154.

DeMers, D., and Cottrell, G. (1993). Non-Linear Dimensionality Reduction. In S. J. Hanson, J. D. Cowan, and C. L. Giles (Eds.), *Advances in Neural Information Processing Systems* (Vol. 5, pp. 580–587). San Mateo, CA: Morgan Kaufmann.

Dempster, A. P., Laird, N. M., and Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society, Series B*, *39*(1), 1–38.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, *1*(1), 269–271.

Donoho, D. L., and Grimes, C. (2003). Hessian eigenmaps: locally linear embedding techniques for high dimensional data. *Proc. of National Academy of Sciences*, *100*(10), 5591–5596.

Drucker, H., Burges, C. J. C., Kaufman, L., Smola, A. J., and Vapnik, V. (1997). Support Vector Regression Machines. In M. C. Mozer, M. I. Jordan, and T. Petsche (Eds.), *Advances in Neural Information Processing Systems* (Vol. 9, pp. 155–161). Cambridge, MA: MIT Press.

Efron, B. (1979). Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*, *7*(1), 1–26.

Efron, B. (1983). Estimating the Error Rate of a Prediction Rule: Improvement on Cross-Validation. *Journal of the American Statistical Association*, *78*(382), 316–

331.

Everitt, B. S. (1984). *An Introduction to Latent Variable Models.* Monographs on Statistics and Applied Probability. London: Chapman and Hall.

Fisher, R. A. (1922). On the Mathematical Foundations of Theoretical Statistics. *Philosophical Transactions of the Royal Society of London, Series A*, *222*, 309–368.

Forster, W. (1995). Homotopy Methods. In R. Horst and P. M. Pardalos (Eds.), *Handbook of Global Optimization.* Nonconvex Optimization and its Applications (Vol. 2). Dordrecht: Kluwer Academic Publishers.

Friedman, J. (1994). An Overview of Predictive Learning and Function Approximation. In V. Cherkassky, J. Friedman, and H. Wechsler (Eds.), *From Statistics to Neural Networks* (Vol. 136 of NATO ASI series F). New York: Springer.

Fyfe, C., and MacDonald, D. (2002). Epsilon-insensitive Hebbian Learning. *Neurocomputing*, *47*, 35–57.

Gersho, A., and Gray, R. M. (1992). *Vector Quantization and Signal Compression.* Boston, MA: Kluwer Academic Publishers.

Gilbert, J. R., Moler, C., and Schreiber, R. (1992). Sparse Matrices in MATLAB: Design and Implementation. *SIAM Journal on Matrix Analysis and Applications*, *13*(1), 333–356.

Green, P. J., and Silverman, B. W. (1994). *Nonparametric Regression and Generalized Linear Models.* Monographs on Statistics and Applied Probability. London: Chapman and Hall.

Hagenbuchner, M., Sperduti, A., and Tsoi, A. C. (2003). A Self-Organizing Map for Adaptive Processing of Structured Data. *IEEE Transactions on Neural Networks*, *14*, 491-505.

Ham, J., Lee, D. D., Mika, S., and Schölkopf, B. (2004). Kernel View of the Dimensionality Reduction of Manifolds. In *Proc. 21st. Int. Conf. on Machine Learning (ICML-04)* (pp. 369–376). Banff, Canada.

Hamerly, G., and Elkan, C. (2002). Alternatives to the K-Means Algorithm that Find Better Clusterings. In *Proc. 11th Int. Conf. on Information and Knowledge Management (CIKM '02)* (pp. 600–607). New York: ACM Press.

Hartigan, J. A. (1975). *Clustering Algorithms.* New York: Wiley.

Hastie, T. (1984). *Principal Curves and Surfaces.* Phd thesis, Stanford University.

Hastie, T., and Stuetzle, W. (1989). Principal Curves. *Journal of the American Statistical Association*, *84*(406), 502–516.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2001). *The Elements of Statistical Learning*. New York: Springer.

Hein, M., and Audibert, J.-Y. (2005). Intrinsic Dimensionality Estimation of Submanifolds in R^d. In *Proc. 22nd Int. Conf. Machine Learning (ICML)* (pp. 289–296). Bonn, Germany: ACM.

Hertz, J., Krogh, A., and Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison-Wesley.

Hinton, G. E., Dayan, P., and Revow, M. (1997). Modelling the Manifolds of Images of Handwritten Digits. *IEEE Transactions on Neural Networks*, *8*, 65–74.

Hinton, G. E., and Salakhutdinov, R. R. (2006). Reducing the Dimensionality of Data with Neural Networks. *Science*, *313*(5786), 504–507.

Hotelling, H. (1933). Analysis of a Complex of Statistical Variables with Principal Components. *Journal of Educational Psychology*, *24*, 417–441.

Huber, P. J. (1981). *Robust Statistics*. New York: Wiley.

Härdle, W. (1990). *Applied Nonparametric Regression*. Cambridge: Cambridge University press.

Härdle, W., and Marron, J. S. (1985). Optimal Bandwidth Selection in Nonparametric Regression Function Estimation. *The Annals of Statistics*, *13*(4), 1465–1481.

Härdle, W., and Simar, L. (2003). *Applied Multivariate Statistical Analysis*. Berlin: Springer.

Igel, C., and Hüsken, M. (2000). Improving the RPROP Learning Algorithm. In H. Bothe and R. Rojas (Eds.), *Proc. of the Second International ICSC Symposium on Neural Computation (NC 2000)* (pp. 115–121). ICSC Academic Press.

Jain, A. K., and Dubes, R. C. (1988). *Algorithms for Clustering Data*. Englewood Cliffs, NJ: Prentice Hall.

Jolliffe, I. T. (2002). *Principal Component Analysis* (2nd ed.). New York: Springer.

Jonathan, P., Krzanowski, W. J., and McCarthy, W. V. (2000). On the Use of Cross-validation to Assess Performance in Multivariate Prediction. *Statistics and Computing*, *10*(3), 209–229.

Kambhatla, N., and Leen, T. K. (1994). Fast Non-linear Dimension Reduction. In J. D. Cowan, G. Tesauro, and J. Alspector (Eds.), *Advances in Neural Information Processing Systems* (Vol. 6, pp. 152–159). San Mateo, CA: Morgan Kaufmann Publishers, Inc.

Kaski, S., Kangas, J., and Kohonen, T. (1998). Bibliography of Self-Organizing Map (SOM) Papers: 1981-1997. *Neural Computing Surveys*, *1*, 102-350.

Kégl, B., Krzyzak, A., Linder, T., and Zeger, K. (2000). Learning and Design of Principal Curves. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, *22*(3), 281–297.

Klanke, S., Lebedev, D., Haschke, R., Steil, J. J., and Ritter, H. (2006). Dynamic Path Planning for a 7-DOF Robot Arm. In *Proc. Int. Conf. on Intelligent Robots and Systems (IROS'06)* (pp. 3879–3884).

Klanke, S., and Ritter, H. (2005). PSOM+: Parametrized Self-Organizing Maps for Noisy and Incomplete Data. In *Proc. 5th Workshop on Self-Organizing Maps (WSOM)* (pp. 363–370). Paris, France.

Klanke, S., and Ritter, H. (2006a). A Leave-K-Out Cross-Validation Scheme for Unsupervised Kernel Regression. In S. Kollias, A. Stafylopatis, W. Duch, and E. Oja (Eds.), *Artificial Neural Networks – ICANN 2006* (Vol. 4132, pp. 427–436). Berlin/Heidelberg: Springer.

Klanke, S., and Ritter, H. (2006b). Variants of Unsupervised Kernel Regression: General Cost Functions. In M. Verleysen (Ed.), *Proc. 14th Eur. Symp. Artificial Neural Networks (ESANN)* (p. 581-586). Evere, Belgium: d-side.

Klanke, S., and Ritter, H. (2007). Variants of Unsupervised Kernel Regression: General Cost Functions. *Neurocomputing*. (doi:10.1016/j.neucom.2006.11.015)

Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In *Proc. 14th Int. Joint Conf. on Artificial Intelligence* (Vol. 2, p. 1137-1145). Montreal: Morgan Kaufmann Publishers, Inc.

Kohonen, T. (1982). Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, *43*, 59–69.

Kohonen, T. (2001). *Self-Organizing Maps* (3rd ed.). Berlin: Springer.

Koikkalainen, P., and Oja, E. (1990). Self-Organizing Hierarchical Feature Maps. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN)* (Vol. II, p. 279-284). Piscataway, NJ: IEEE Service Center.

Kramer, M. A. (1991). Nonlinear Principal Component Analysis using Autoassocia-

tive Neural Networks. *Journal for the American Institute of Chemical Engineers (AICHE)*, *37*, 233–243.

Kwok, J. T., and Tsang, I. W. (2002). Finding the pre images in kernel principal component analysis. In *6th Annual Workshop On Kernel Machines.* Whistler.

LeBlanc, M., and Tibshirani, R. (1994). Adaptive principal surfaces. *Journal of the American Statistical Association*, *89*(425), 53–64.

Lee, J. A., Archambeau, C., and Verleysen, M. (2003). Locally Linear Embedding versus Isotop. In M. Verleysen (Ed.), *Proc. 11th Eur. Symp. Artificial Neural Networks (ESANN).* Evere, Belgium: d-side.

Lee, J. A., Lendasse, A., Donckers, N., and Verleysen, M. (2000). A Robust Non-linear Projection Method. In M. Verleysen (Ed.), *Proc. 8th Eur. Symp. Artificial Neural Networks (ESANN)* (pp. 13–20). Evere, Belgium: d-side.

Lee, J. A., Lendasse, A., and Verleysen, M. (2004). Nonlinear Projection with Curvilinear Distances: Isomap versus Curvilinear Distance Analysis. *Neurocomputing*, *57*, 49–76.

Lee, J. A., and Verleysen, M. (2002). Nonlinear Projection with the Isotop Method. In J. Dorronsoro (Ed.), *Artificial Neural Networks – ICANN 2002* (Vol. 2415, pp. 933–938). Berlin/Heidelberg: Springer.

Lendasse, A., Simon, G., Wertz, V., and Verleysen, M. (2005). Fast Bootstrap Methodology for Regression Model Selection. *Neurocomputing*, *64*, 161-181.

Lendasse, A., Wertz, V., and Verleysen, M. (2003). Model Selection with Cross-Validations and Bootstraps – Application to Time Series Prediction with RBFN Models. In O. Kaynak, E. Alpaydin, E. Oja, and L. Xu (Eds.), *Artificial Neural Networks and Neural Information Processing – ICANN/ICONIP* (Vol. 2714, pp. 573–580). Berlin/Heidelberg: Springer.

Leung, D. H.-Y. (2005). Cross-validation in Nonparametric Regression with Outliers. *Annals of Statistics*, *33*(5), 2291-2310.

Linde, Y., Buzo, A., and Gray, R. M. (1980). An Algorithm for Vector Quantizer Design. *IEEE Transactions on Communications*, 702–710.

Lindsay, B. G. (1983). The Geometry of Mixture Likelihoods: A General Theory. *Annal of Statistics*, *11*, 86–94.

MacQueen, J. B. (1967). Some Methods for Classification and Analysis of Multivariate Observations. In *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability* (Vol. 1, pp. 281–297). Berkeley, CA: University of California Press.

Mahalanobis, P. C. (1957). *Statistics*. Paris: Unesco.

Martinetz, T., Berkovich, S., and Schulten, K. (1993). "Neural-Gas" Network for Vector Quantization and its Application to Time-Series Prediction. *IEEE-Transactions on Neural Networks*, *4*(4), 558–1569.

Martinetz, T., and Schulten, K. (1991). A "Neural-Gas" Network Learns Topologies. *Artificial Neural Networks*, *I*, 397–402.

Meinicke, P. (2000). *Unsupervised Learning in a Generalized Regression Framework*. PhD thesis, University of Bielefeld.

Meinicke, P., Klanke, S., Memisevic, R., and Ritter, H. (2005). Principal Surfaces from Unsupervised Kernel Regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(9), 1379–1391.

Mercer, J. (1909). Functions of Positive and Negative Type, and Their Connection with the Theory of Integral Equations. *Proc. of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, *83*(559), 69–70.

Moré, J. J., and Wu, Z. (1997). Global Continuation for Distance Geometry Problems. *SIAM Journal on Optimization*, *7*(3), 814–836.

Möller, R., and Hoffmann, H. (2004). An Extension of Neural Gas to Local PCA. *Neurocomputing*, *62*, 305–326.

Möller, R., and Könies, A. (2004). Coupled Principal Component Analysis. *IEEE Transactions on Neural Networks*, *15*(1), 214–222.

Nadaraya, E. A. (1964). On Estimating Regression. *Theory of Probability and Its Application*, *9*, 141–142.

Nölker, C., and Ritter, H. (2000). Parametrized SOMs for Hand Posture Reconstruction. In S. I. Amari, C. L. Giles, M. Gori, and V. Piuri (Eds.), *Proc. Int. Joint Conf. on Neural Networks* (pp. 139–144).

Oja, E. (1982). A Simplified Neuron Model as a Principal Component Analyzer. *Journal of Mathematical Biology*, *15*, 267–273.

Oja, M., Kaski, S., and Kohonen, T. (2003). Bibliography of Self-Organizing Map (SOM) Papers: 1998-2001 Addendum. *Neural Computing Surveys*, *3*, 1-156.

Ontrup, J., and Ritter, H. (2005). A Hierarchically Growing Hyperbolic Self-Organizing Map for Rapid Structuring of Large Data Sets. In *Proc. 5th Workshop on Self-Organizing Maps (WSOM)* (p. 471-478). Paris, France.

Otten, R. H. J. M., and van Ginneken, L. P. P. P. (1992). *The Annealing Algorithm*. Boston, MA: Kluwer Academic Publishers.

Padoan Jr., A. C., Barreto, G. A., and Araújo, A. F. R. (2003). Modeling and production of robot trajectories using the temporal parametrized self-organizing map. *International Journal of Neural Systems*, *13*(2), 119–127.

Parzen, E. (1962). On the estimation of a probability density function and mode. *Annals of Mathematical Statistics*, *33*, 1065–1076.

Pearson, K. (1901). On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine and Journal of Science*, *2*, 559–572.

Peña, M., and Fyfe, C. (2006). Outlier identification with the harmonic topographic mapping. In *Proc. european symposium on artificial neural networks* (pp. 289–294).

Quist, M., and Yona, G. (2004). Distributional Scaling: An Algorithm for Structure-Preserving Embedding of Metric and Nonmetric Spaces. *Journal of Machine Learning Research*, *5*, 399–420.

Rao, C. R. (1964). The Use and Interpretation of Principal Component Analysis in Applied Research. *Sankhya Series A*, *26*, 329–358.

Rao, S. S. (1978). *Optimization: Theory and Applications*. New Delhi: Wiley Eastern Ltd.

Riedmiller, M., and Braun, H. (1993). A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP algorithm. In *Proc. of the IEEE Int. Conf. on Neural Networks* (pp. 586–591).

Ripley, B. D. (2005). *Pattern Recognition and Neural Networks*. Cambridge: Cambridge University Press.

Ritter, H. (1993). Parametrized Self-Organizing Maps. In S. Gielen and B. Kappen (Eds.), *Proc. Int. Conf. Artificial Neural Networks (ICANN)* (pp. 568–577). London, UK: Springer.

Ritter, H. (1999). Self-Organizing Maps in Non-Euclidean Spaces. In E. Oja and S. Kaski (Eds.), *Kohonen Maps* (pp. 97–108).

Ritter, H., Martinetz, T., and Schulten, K. (1992). *Neural Computation and Self-Organizing Maps*. Addison Wesley.

Rose, K., Gurewitz, E., and Fox, G. (1990). A Deterministic Annealing Approach to Clustering. *Pattern Recognition Letters*, *11*(9), 589–594.

Roweis, S. T., and Saul, L. K. (2000). Nonlinear Dimensionality Reduction by Locally Linear Embedding. *Science*, *290*, 2323–2326.

Ruiz de Angulo, V., and Torras, C. (2002). Learning inverse kinematics via cross-point function decomposition. In J. Dorronsoro (Ed.), *Artificial Neural Networks – ICANN 2002* (Vol. 2415, pp. 856–864). Berlin/Heidelberg: Springer.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning Internal Representations by Error Propagation. In D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* (Vol. 1: Foundations, pp. 318–362). Cambridge, MA: MIT Press.

Saalbach, A., Heidemann, G., and Ritter, H. (2002). Parametrized SOMs for Object Recognition and Pose Estimation. In J. Dorronsoro (Ed.), *Artificial Neural Networks – ICANN 2002* (Vol. 2415, pp. 902–907). Berlin/Heidelberg: Springer.

Sammon, J. W., Jr. (1969). A Non-linear Mapping for Data Structure Analysis. *IEEE Transactions on Computers*, *18*, 401–409.

Sanger, T. (1989). Optimal Unsupervised Learning in a Single-layer Linear Feedforward Neural Network. *Neural Networks*, *12*, 459–473.

Saul, L. K., and Roweis, S. T. (2003). Think Globally, Fit Locally: Unsupervised Learning of Low Dimensional Manifolds. *Journal of Machine Learning Research*, *4*, 119–155.

Saul, L. K., Weinberger, K. Q., Ham, J. H., Sha, F., and Lee, D. D. (2006). Spectral Methods for Dimensionality Reduction. In O. C. B. Schölkopf and A. Zien (Eds.), *Semisupervised learning.* MIT Press.

Schölkopf, B., and Smola, A. J. (2002). *Learning with Kernels.* Cambridge, MA: MIT Press.

Schölkopf, B., Smola, A. J., and Müller, K.-R. (1998). Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, *10*, 1299–1319.

Scott, D. W. (1992). *Multivariate Density Estimation.* New York: Wiley.

Shepard, R. N. (1980). Multidimensional Scaling, Tree-Fitting, and Clustering. *Science*, *210*(4468), 390–398.

Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis.* Monographs on Statistics and Applied Probability. London: Chapman and Hall.

Smola, A. J., Mika, S., Schölkopf, B., and Williamson, R. C. (2001). Regularized Principal Manifolds. *Journal of Machine Learning Research*, *1*, 179–209.

Smola, A. J., Schölkopf, B., and Müller, K. (1998b). General Cost Functions for Support Vector Regression. In T. Downs, M. Frean, and M. Gallagher (Eds.), *Proc. 9th Australian Conf. on Neural Networks* (pp. 79–83). Brisbane, Australia: University of Queensland.

Smola, A. J., Schölkopf, B., and Müller, K.-R. (1998a). The connection between regularization operators and support vector kernels. *Neural Networks*, *11*(4), 637–649.

Smolensky, P. (1986). Information Processing in Dynamical Systems: Foundations of Harmony Theory. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations* (pp. 194–281). Cambridge, MA: MIT Press.

Stone, C. J. (1977). Consistent Nonparametric Regression. *The Annals of Statistics*, *5*(4), 595–620.

Stone, M. (1974). Cross-Validatory Choice and Assessment of Statistical Predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, *36*(2), 111–147.

Stone, M. (1977). An Asymptotic Equivalence of Choice of Model by Cross-Validation and Akaike's Criterion. *Journal of the Royal Statistical Society. Series B (Methodological)*, *39*(1), 44–47.

Tenenbaum, J. B., Silva, V. de, and Langford, J. C. (2000). A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, *290*, 2319–2323.

Tibshirani, R. (1992). Principal Curves Revisited. *Statistics and Computing*, *2*, 183–190.

Tipping, M. E., and Bishop, C. M. (1997). *Probabilistic Principal Component Analysis* (Technical Report NCRG/97/010, Neural Computing Research Group). Birmingham, UK: Aston University.

Tipping, M. E., and Bishop, C. M. (1999a). Mixtures of Probabilistic Principal Component Analysers. *Neural Computation*, *11*(2), 443–482.

Tipping, M. E., and Bishop, C. M. (1999b). Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society, Series B*, *61*, 611–622.

Utsugi, A. (1997a). Hyperparameter Selection for Self-Organizing Maps. *Neural Computation*, *9*(3), 623-635.

Utsugi, A. (1997b). Topology Selection for Self-Organizing Maps. *Network: Computation in Neural Systems*, *7*, 727-740.

Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory.* New York: Springer.

Vapnik, V. N. (1998). *Statistical Learning Theory.* New York: Wiley.

Verbeek, J. J., Vlassis, N., and Kröse, B. (2001). A Soft K-Segments Algorithm for Principal Curves. In G. Dorffner, H. Bischof, and K. Hornik (Eds.), *Artificial Neural Networks – ICANN* (Vol. 2130, p. 450-456). Berlin/Heidelberg: Springer.

Verbeek, J. J., Vlassis, N., and Kröse, B. (2002). A K-Segments Algorithm for Finding Principal Curves. *Pattern Recognition Letters*, *23*, 1009-1017.

Vijayakumar, S., and Schaal, S. (2000). Locally Weighted Projection Regression: An O(n) Algorithm for Incremental Real Time Learning in High Dimensional Space. In *Proc. Int. Conf. on Machine Learning (ICML)* (pp. 1079–1086). San Francisco, CA: Morgan Kaufmann Publishers, Inc.

Vishwanathan, S. V. N., and Smola, A. J. (2003). Fast Kernels on Strings and Trees. In *Advances in Neural Information Processing Systems* (Vol. 15). Cambridge, MA: MIT Press.

Walter, J. (1998). PSOM Network: Learning with Few Examples. In *Proc. Int. Conf. on Robotics and Automation (ICRA)* (pp. 2054–2059).

Walter, J., Nölker, C., and Ritter, H. (2000). The PSOM Algorithm and Applications. In *Proc. Symposion Neural Computation* (pp. 758–764).

Walter, J., and Ritter, H. (1995). Local PSOMs and Chebyshev PSOMs – Improving the Parametrised Self-Organizing Maps. In *Proc. Int. Conf. Artificial Neural Networks (ICANN)* (Vol. 1, pp. 95–102).

Walter, J., and Ritter, H. (1996). Rapid Learning with Parametrized Self-Organizing Maps. *Neurocomputing*, *12*, 131–153.

Wang, F. T., and Scott, D. W. (1994). The l_1 Method for Robust Nonparametric Regression. *Journal of the American Statistical Association*, *89*(425), 65–76.

Watson, G. S. (1964). Smooth Regression Analysis. *Sankhya, Series A*, *26*, 359–372.

Weinberger, K. Q., Packer, B. D., and Saul, L. K. (2005). Nonlinear Dimensionality Reduction by Semidefinite Programming and Kernel Matrix Factorization. In *Proc. 10th Int. Workshop on Artificial Intelligence and Statistics.* Barbados.

Weinberger, K. Q., and Saul, L. K. (2004). Unsupervised Learning of Image Manifolds by Semidefinite Programming. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR-04)* (Vol. 2, pp. 988–995). Washington D.C.

Weinberger, K. Q., and Saul, L. K. (2006). Unsupervised Learning of Image Manifolds by Semidefinite Programming. *International Journal of Computer Vision*, *70(1)*,

77–90.

Weinberger, K. Q., Sha, F., and Saul, L. K. (2004). Learning a Kernel Matrix for Nonlinear Dimensionality Reduction. In *Proc. 21st Int. Conf. on Machine Learning (ICML-04)* (pp. 839–846). Banff, Canada.

Weston, J., Chapelle, O., Elisseeff, A., Schölkopf, B., and Vapnik, V. (2003). Kernel dependency estimation. In *Advances in Neural Information Processing Systems* (Vol. 15). Cambridge, MA: MIT Press.

Yang, L., and Tschernig, R. (1999). Multivariate Bandwidth Selection for Local Linear Regression. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, *61*(4), 793–815.

Yuan, M., and Wahba, G. (2004). *Doubly Penalized Likelihood Estimator in Heteroscedastic Regression* (Technical Report No. 1084rr). Department of Statistics, University of Wisconsin, Madison WI.

Zhang, B. (2000). *Generalized K-Harmonic Means – Boosting in Unsupervised Learning* (Technical Report HPL-2000-137). Palo Alto: Hewlett-Packard Laboratories.

Zhang, B., Fu, M., and Yan, H. (1998). Handwritten Digit Recognition by a Mixture of Local Principal Component Analysis. *Neural Processing Letters*, *8*, 241–252.

Zhang, B., Hsu, M., and Dayal, U. (1999). *K-Harmonic Means – A Data Clustering Algorithm* (Technical Report HPL-1999-124). Palo Alto: Hewlett-Packard Laboratories.