Extensions to Kernel Dependency Estimation

— with Applications to Robotics —

vorgelegt von
Diplom Ingenieur
Gökhan Hasan Bakır
aus Istanbul

von der Fakultät IV
Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr. Ing. -
genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. G. Hommel
Berichter:	Prof. Dr. B. Jähnichen
Berichter:	Prof. Dr. B. Schölkopf

Tag der wissenschaftlichen Aussprache: 15. Dezember 2005

Berlin 2006
D 83

Fatma Meral Bakır, Evrim Ergül-Bakır

ve

kendim

için

Contents

Foreword

Kernel Dependency Estimation (KDE) is a recently introduced technique that learns general functional dependencies between structured input and output data. The general framework introduced by KDE, of dealing with structured data by employing an embedding step, a mapping step, and solving a pre-image problem to invert the mapping allows many possible implementations.

The subject of this book is the exploration of such methods, from both a theoretic and algorithmic point of view, without losing sight of a target application of these methods in the field of robotics. This includes the analysis of the interplay between embeddings, mappings and loss functions, the introduction of the algorithm Multivariate regression via Stiefel constraints, and strategies for efficiently solving the pre-image problem for vectors, graphs and sequences.

The second part of the book, in Chapters 5 and 6 presents a mathematical approach to the problem of pose imitation (from human to robot). The author then goes on to present a novel learning approach utilizing all the developed tools in this thesis to give an efficient imitation learning system which can operate in the presence of only limited domain knowledge. The potential of such a system, which combines kernel design from image processing through to kinematics, advanced regression techniques and constrained pre-image algorithms is an exciting prospect, from my point of view because many of the components are well motivated general techniques rather than heuristics. While being theoretically motivated, the approach never loses sight of the use of domain knowledge when applicable, which I find quite an achievement.

I find this work, submitted for the title of *Extensions to Kernel Dependency Estimation*, a substantial contribution to Machine Learning. I certainly hope the reader does as well.

Jason Weston, NEC Research Labs America, Princeton, USA

Acknowledgment

This thesis is the possible result of the Brownian motion of my life.

First of all, I am thankful for Prof. Bernhard Schölkopf giving me the opportunity to enter his lab and pursue a PhD in his group. Bernhard was always available for discussion of fresh and mostly crazy ideas. Furthermore, I would like to thank Prof. Jähnichen for accepting this thesis.
I have expected a challenging environment among high motivated students but was pleased to find a motivating (still challenging) environment among future friends. Furthermore, I am indebted to Matthias Otto Franz who actually accepted me as his Ph.D. student. He mainly teached and guided me how to approach problems in a systematic way, think and write scientifically and stop breeding chaotic thoughts. Whenever I was jumping up with a new exciting idea he was ready to share my thoughts and required me to reconsider details always ready to give a crucial hint.

I am also grateful to met my second supervisor Jason Weston which mainly introduced me to the idea of Kernel Dependency Estimation. All our cooperation mostly ended up in me being astonished about how easy Jason was always able to explain and discuss topics. During my PhD time I had the chance to stay at the NEC Laboratories in Princeton, NJ where I had the chance to work with Léon Bottou, Vladimir Vapnik and Jason. At that time I made another quantum leap in understanding kernel methods and especially Support Vector Machines. Thank you!

I had the pleasure to learn or work with many people during my PhD. Without these interactions, parts of this thesis would not have been possible. In the first place I am thankful to Martin Giese, Arthur Gretton, Olivier Chapelle, Alexander Zien, Koji Tsuda, Olivier Bousquet, André Elisseeff and Wolf Kienzle. Furthermore I am grateful to all my fellow PhD colleagues for sharing a good time in the lab. Here, especially I would like to thank Matthias Hein, Jan Eichhorn, Ulrike von Luxburg and Malte Kuß.

I would never have reached this point without the support of Bernd Gombert which I owe more than I could express here. Furthermore, there were several people which strongly influenced me during my time at Logitech and the DLR. One person was Volker Senft which always challenged every academical reasoning for practical applicability. I tried to adopt his healthy skepticism for my scientific attitude. The other main mentor at that time was Holger Weiss, who mainly motivated me to reconsider my whole life and was one of the main reasons to leave industry and start the Ph.D. program at the Max-Planck Institute.

Finally I would like to thank my family and my wife for all their support. Thank you all!

Notation and Symbols

In this thesis the following list of symbols and abbreviations is used unless stated otherwise.

$\mathcal{F}_z$	Reproducing Kernel Hilbert Space associated with kernel z
ϕ_z	Feature map $\phi_z : \mathcal{X}/\mathcal{Y} \to \mathcal{F}_z$
$\mathcal{X}$	Input domain
$\mathcal{Y}$	Output domain
$\mathcal{Z}$	Product space of $\mathcal{X}$ and $\mathcal{Y}$
o	Dimension of output space $\dim \mathcal{Y}$
d	Dimension of input space $\dim \mathcal{X}$
$T_{\mathcal{F}}$	Mapping between Reproducing Kernel Hilbert Spaces
$t_{\mathcal{Z}}$	Mapping between sets $\mathcal{X}$ and $\mathcal{Y}$
D_N	Training set consisting of N points $\{x_i, y_i\}_{i=1}^N$ or just $\{x_i\}_{i=1}^N$

Chapter 1

Introduction and Preliminaries

The fundamental goal in robotics is to build machines which are smart and versatile enough to act in as diverse fields as a human is able to act. A key ability of humans which any robot lacks is to learn from example behavior observed from humans. This ability is called *imitation learning*. A particular key problem to imitation learning is the ability to infer the state of an actor in the perceived environment and being able to map perceived informations into own actions. In particular we will focus on the scenario where a human actor is moving his right arm while a camera is watching the human actor. A computer analyzes the human pose and has to command a robot manipulator with seven degrees of freedom to a state which resembles the observed posture as shown in figure 1.1. In this thesis we treat this particular problem as a single approximation problem where our imitation map has the form

$$t : \text{Observation of Human Action} \rightarrow \text{State of Robot.}$$

The problem with such a formulation is that the imitating map is not necessarily a function in the usual sense since the output map is not necessarily a vector space. Thus we consider the problem of estimating a map between arbitrary input and output sets. To this end, we investigate and extend an existing machine learning algorithm based on kernel functions – Kernel Dependency Estimation (KDE) [99] – which is capable of learning in this general setup.

The purpose of the following sections in this chapter is to give the necessary background material for the theoretical concepts used. In the first section, we discuss relevant concepts of learning and regularization, whereas in the succeeding section we introduce the concept of kernels and feature spaces which are crucial for the application of KDE. In the final section in this chapter we review the KDE algorithm in detail and give an overview of the thesis.

1.1 Supervised Learning

A key step in building robots which can act in an unknown environment is the ability to perform predictions of future states from past observations. In this thesis we utilize the paradigm of supervised learning meaning that an external *supervisor* presents a set of examples which consist of input patterns and corresponding output patterns. It is expected that the learner *generalizes* to inputs it did not see before. Let us formalize this. In supervised learning a learning algorithm $\mathcal{A}$ is given a set D_N of example pairs of inputs $x_i \in \mathcal{X}$ and desired outputs $y_i \in \mathcal{Y}$ and a set H of possible hypotheses. The task of the learning algorithm is now to choose a hypothesis $t \in H$ according to an *induction* principle.

Figure 1.1: Our robot while imitating a human pose.

For example, a straightforward induction principle is to choose the best hypothesis t which explains the given examples perfectly:

$$t \in \{t | t(x_i) = y_i\}.$$

But, since we are interested in a generalization performance after a *teach in* phase we are not interested in a hypothesis which predicts best on the training inputs but on other unseen examples. To this end, let us introduce a cost function l which will serve us in assessing the performance of an individual hypothesis. We define the cost function as

$$l(x, y, \hat{y}) : \mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}_+,$$

and its task is to measure the cost of mismatch between predictions $\hat{y} := t(x)$ and the true y, given a particular data pair $(x, y) \in \mathcal{Z}$ and $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$. Trivially, the best generalizing hypothesis t^* could be chosen if one just would have knowledge over *all possible* future examples, i.e. if one would know the joint distribution function $\mathbb{P}(x, y)$. In this case minimizing the *expected* loss is guaranteed to be the best choice:

$$t^* = \arg\min_{t \in H} R_{\mathbb{P}}[t] := \arg\min_{t \in H} \mathbb{E}_{(x,y)\sim\mathbb{P}(x,y)} l\left(x, y, t(x)\right).$$

Thus, if we know the underlying distribution $\mathbb{P}(x)$ of all future inputs x and the conditional distribution $\mathbb{P}(y|x)$ of the outputs y we would be able to construct the statistically *optimal* hypothesis. Unfortunately, this is rarely the case in practice since we have only a finite set of examples D_N and thus a limited knowledge of the joint distribution $\mathbb{P}(x, y)$. Nevertheless we can use the *Empirical Risk Minimization*(ERM) induction principle which chooses a hypothesis that yields the minimum empirical loss on D_N:

$$t_{emp} = \arg\min_{t \in H} R_{emp}[t, D_N] := \arg\min_{t \in H} \frac{1}{N} \sum_{i=1}^{N} l(x_i, y_i, t(x_i)). \tag{1}$$

Fortunately, due the following theorem T1.1 by V. Vapnik [95] we have the guarantee that t_{emp} will approach t^* with an increasing number of examples: [1]

[1] We assume that our hypothesis class H consists of hypotheses such that all integrals of the form $\sup_{t \in H} R_{\mathbb{P}}[t] < \infty$ are finite.

Theorem T1.1 ERM is consistent. Vapnik [95, chapter 3]
Let $R_{\mathbb{P}}[t]$ *be the expected* risk

$$R_{\mathbb{P}}[t] = \mathbb{E}_{(x,y)\sim\mathbb{P}(x,y)} l(x, y, t(x)),$$

which measures the average cost over all possible *predictions the hypothesis has to perform. Given the empirical estimate* $R_{emp}[t, D_N]$ *of* $R_{\mathbb{P}}[t]$ *where* D_N *is a finite sample from* $\mathbb{P}(x, y)$ *the following holds for all* $\epsilon > 0$*:*

$$\lim_{N\to\infty} \mathbb{P}\left(\sup_{t\in H} |R_{emp}[t, D_N] - R_{\mathbb{P}}[t]| > \epsilon\right) = 0. \tag{2}$$

The theorem tells us that $R_{emp}[t, D_N]$ converges to $R_{\mathbb{P}}[t]$ even in the worst case with increasing N and therefore is of *asymptotic* nature. For a finite number of samples however the empirical measured risk R_{emp} is almost surely lower than the expected risk [95]. Indeed, it can happen that a chosen hypothesis has empirical risk zero (our model explains the finite set of observations perfectly), but the average prediction error for all expected inputs is larger than zero. An analysis of upper bounds on the expected risk, as was done by Vapnik [95], reveals that another quantity is crucial for generalization: The capacity of H. The capacity of a hypotheses set H can be understood as a measure for the amount of data which can be explained by some hypothesis in this set. For example, a valid capacity measure is: What is the size of the smallest data set D_N which will lead to $\inf_t R_{emp}[t, D_N] > 0$? Or in plain words: What is the size of the smallest data set D_N that I can not explain anymore with any of my available hypotheses in the set H. [2] Now almost all upper bounds which uses the concept of capacity have the form:

$$R_{\mathbb{P}} \leq R_{emp}[t, D_N] + \frac{1}{\sqrt{N}} C(H),$$

where the first term on the right side is the empirical risk and the second term is some measure on the capacity of H. If the amount of training data is large, the confidence in choosing a hypothesis based on the empirical risk is high and the influence of the capacity might be negligible. On the other side, if the amount of training data is low and the capacity of the used hypothesis set H is big, it might happen that by accident a hypothesis in H explains only the training data well. This phenomenon is known as *overfitting* and is of central importance in machine learning. To avoid overfitting one usually accepts a trade-off in explaining observations in D_N and exploring the set of all possible hypothesis. This is the main concern of *regularization* which we discuss next.

1.1.1 Regularization

The key idea in regularization can be formulated as incorporation of a priori assumptions about the desired solution t to *control* the gap between R_{emp} and $R_{\mathbb{P}}$ and thus to control the exploitation of the hypothesis set H. To this end one extends the minimization problem in (1) with a so called *sympathy* functional[39] $\Omega[t] : H \to \mathbb{R}$, which encodes a preference for solutions. An example how to extend the minimization problem is by adding $\Omega[t]$ to obtain the new minimization approach

$$h = \arg\min_{t\in H} R_{emp}[t, D_N] + \Omega[t],$$

[2] For some possible capacity measures used in learning theory, see for example [95].

or alternatively to use it as a constraint

$$\begin{aligned} t &= \arg\min_{t \in H} R_{emp}[t, D_N] \\ &\quad \text{subject to } \Omega[t] \leq \eta, \end{aligned}$$

provided a level parameter $\eta \in \mathbb{R}$. We will encounter the latter form of regularization in chapter 2. As an example for a sympathy functional $\Omega[t]$ consider the distance to a preferred skeleton solution t_0 (for a detailed discussen see e.g. Hofmann [39]) [3]

$$\Omega_{t_0}[t] = \lambda ||t - t_0||_H^2, \tag{3}$$

or for example if t is a function then $\Omega[t]$ can penalize properties of t such as smoothness or variation [39] , e.g.:

$$\Omega_n[t] = \lambda \sum_{i=1}^{n} ||\frac{\partial^i}{\partial x^i} h||_H^2. \tag{4}$$

The coefficient $\lambda \in \mathbb{R}$ in (3),(4) is used to control the trade-off between fit and sympathy functional and is mostly determined by cross-validation [39]. A simple extension to (3) is the widely used *minimum norm* regularization functional where the preferred solution t_0 is zero and thus the regularizer is of the form

$$\Omega_0[t] = \lambda ||t||_H^2. \tag{5}$$

The minimum norm functional is the most studied regularizer in the literature. In the next section we will see a geometric interpretation for the case of classification and discuss why the addition of minimum norm sympathy functional controls the gap between R_{emp} and $R_{\mathbb{P}}$.

In contrast to extending the minimization problem with a regularization functional, a different, quite algorithmic approach is to stop before finishing minimization. This approach, called *early stopping* is widely used in the neural network community and aims also to perturb the minimization of R_{emp} but more in a rather uncontrolled manner. In chapter 2 we will encounter an algorithm called Partial Least Squares which uses an early stopping approach. Since early stopping is an uncontrolled regularization mechanism we will propose an alternative based on other regularization functionals. In the following two sections we will give in the following two concrete examples for supervised learning tasks: The task of classification and regression.

1.1.2 Classification

In the section we will restrict $\mathcal{X}$ to be a real d-dimensional vector space $\mathbb{R}^d$. The task of classification is to construct a hypothesis when the output is restricted to 1 or -1 denoting a yes/no decision. For example the task to *detect* a human in the image can be posed as a classification problem. Thus, we are given a set of input vectors $x_i \in \mathbb{R}^d$ and their corresponding class information $y_i \in \{1, -1\}$ as training data set $D_N = \{x_i, y_i\}_{i=1}^N$. A natural loss function l in this setting is the zero-one or yes/no loss

$$l_{0/1}(x, y, t) = \begin{cases} 1 & \text{if } y \neq t(x) \\ 0 & \text{else} \end{cases}$$

[3] For example as in the Kalman filtering algorithm where t_0 is the previous solution. This ensures a smooth change of t over time.

For the ideal case that we have a statistical model of all possible inputs $\mathbb{P}(x)$ with the corresponding label information $\mathbb{P}(y|x)$ one can show that the the optimal classifier for the zero-one loss function is the so called *Bayes* classifier t_{Bayes} which is given by:

$$t_{Bayes}(x) = \begin{cases} +1 & \text{if} \quad \mathbb{P}(y=+1|x)\mathbb{P}(x) > \mathbb{P}(y=-1|x)\mathbb{P}(x) \\ -1 & \text{else} \end{cases}$$

Practically, since one does not have these statistical informations one has to use the ERM principle together with a regularization scheme. Before we discuss how the ERM principle can be implemented for classification in detail, we have to agree upon a hypothesis class H. In this thesis, we will be mainly concerned with the class of linear hypotheses, i.e.:

$$H = \left\{ (w,b) | w^\top x + b, \text{ where } (w,b) \in \mathbb{R}^{d+1} \right\},$$

and thus the final hypotheses for classification takes the form $t(x) = \text{sign}(w^\top x + b)$. Having agreed on the hypothesis class, how can we implement ERM for classification? It turns out that this is a very difficult task since minimizing the zero-one loss gives rise to a discrete optimization problem. Therefore, we need to use a loss function being more amenable to optimization. For example, in [95] it was suggested to use the linear loss function:

$$l_1(x,y,t) = \begin{cases} 1 - y(w^\top x + b) & \text{if } y(w^\top x + b) < 1 \\ 0 & \text{else} \end{cases}$$

A modern classification algorithm which can be interpreted as using this linear loss in combination with minimum norm regularization is the support vector machine (SVM) [95]. [4] The crucial point for SVM is that minimizing the norm leads to maximizing the margin ρ between the classes and the decision surface. This can be seen readily from the prediction equation. Denote by $d_i = w^\top x_i + b$ the distance of point x_i to the (linear) decision surface. The gap between the classes is obviously inverse proportional to this distance and thus to the norm of w. Now, to be useful across different models w_1, w_2 of possibly different scale, we need to declare that the closest points $\tilde{x}_i$ to the decision surface should have distance 1, i.e. $\min_i d_i = 1$. Given this common scale for two models w_1 and w_2 we can compare two models according their implied gap. A larger gap (and thus a hyperplane with smaller norm) is favorable since a model with smaller norm leads to a bigger class gap and therefore is likely to be more robust to variation in the data. This, geometrically quite convincing principle is based on the following fact due to V. Vapnik [95]:

Theorem T1.2 Radius Margin Error Bound
Consider the set H of linear hypotheses, with $t \in H = sign(w^\top x + b)$ and $||w|| \leq 1$, and assume that the data have compact support, i.e. $||x|| \leq R$, $\forall x$. For all distributions $\mathbb{P}$ generating the data, with probability at least $1-\delta$ the probability that a test pattern drawn from $\mathbb{P}$ will be misclassified is bounded from above, by

$$R[t] \leq R_{emp}[t] + \sqrt{\frac{c}{N}\left(\frac{R^2}{\rho^2}\ln\ln N + \ln(1/\delta)\right)},$$

where c is a constant, and N is the number of training patterns.

[4] Note that the original term for complexity control used in [95] is structured risk minimization (SRM). The idea of SRM is to construct a sequence of nested hypothesis classes $H_1 \subset H_2 \subset \cdots \subseteq H$ of increasing complexity, and if two solutions t_i, t_j in H_i and H_j respectively perform equally well, choose the hypothesis which belong to the set of lower complexity.

Thus, for SVMs one has to find a trade-off between minimizing the training error (the first term of the upper bound in T1.2) and maximizing the margin (the second term of the upper bound). This can be formulated as the following optimization problem

$$(w^*, b^*) = \underset{w,b,\xi}{\arg\min} \quad \overbrace{\frac{1}{2}||w||^2}^{\text{Minimize Norm! / Maximize Margin!}} \tag{6}$$
$$\text{subject to} \quad \overbrace{y_i\left(w^\top x_i + b\right) \geq 1}^{\text{Classify Correct!}}, \quad 1 \leq i \leq n.$$

Using additional slack variables $\xi_i \in \mathbb{R}^+$ for each point x_i one can extend (7) to the non-separable case:

$$(w^*, b^*) = \underset{w,b,\xi}{\arg\min} \quad \overbrace{\frac{1}{2}||w||^2}^{\text{Minimize Norm / Maximize Margin!}} + \overbrace{\lambda \sum_{i=1}^{n} \xi_i}^{\text{Minimize } l_1 \text{ loss!}} \tag{7}$$
$$\text{subject to} \quad \overbrace{y_i\left(w^\top x_i + b\right) \geq 1 - \xi_i}^{\text{Classify as correct as possible!}}, \quad 1 \leq i \leq n,$$

where $\lambda > 0$ is the regularization coefficient controlling the trade-off between minimizing the empirical error and maximizing the margin. An alternative way to write (6) and (7) is based on the so called *dual* formulation (see e.g. [24]) of a convex optimization problem. Let us give the description of the dual formulation of (7) in algorithm A1.1. In the case of (7), the dual can be easily derived by forming the Lagrangian

$$L(w, b, \xi, \alpha, \nu) = \frac{1}{2}||w||^2 + \lambda \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i \left(y_i \left(w^\top x_i + b\right) - 1 + \xi_i\right) - \sum_{i=1}^{N} \nu_i \xi_i.$$

The optimal solution can be obtained by considering the critical points of L (for a detailed derivation, see for example [78]). In contrast to the dual formulation, the problem (7) is also called the *primal* formulation.
Whenever we face a classification problem in this thesis, we will use the SVM algorithm. Let us now proceed to the complement to classification: The case of regression.

Algorithm A1.1 LINEAR SUPPORT VECTOR MACHINE

Given N pairs $\{(x_1, y_1), \ldots, (x_N, y_N)\} \in \{\mathbb{R}^d \times \{-1, +1\}\}^N$ and a regularization coefficient $\lambda \in \mathbb{R}$ the support vector machine (SVM) algorithm generates a classifier $\boldsymbol{w}^* = \sum_{i=1} \alpha_i^* x_i$ with $\boldsymbol{\alpha}^*$ obtained by the following optimization problem:

$$\boldsymbol{\alpha}^* = \underset{\alpha \in \mathbb{R}^N}{\arg\min} \ \frac{1}{2}\boldsymbol{\alpha}^\top H \boldsymbol{\alpha} - \boldsymbol{\alpha}^\top \mathbf{1} \tag{8}$$
$$\text{subject to } 0 \leq \boldsymbol{\alpha} \leq \lambda \quad \text{and } y^\top \boldsymbol{\alpha} = 0 \tag{9}$$

where $H = \text{diag}[Y] K \text{diag}[Y]$, $K_{ij} = x_i^\top x_j$ and diag$[Y]$ is a diagonal matrix with diagonals equal to y_i with $i, j = 1, \ldots, N$ and $\lambda > 0$.

1.1.3 Regression

The task of regression is to construct a hypothesis when the output is considered to be real with $\mathcal{Y} \equiv \mathbb{R}^o, o \geq 1$. For the case that $o = 1$, the regression problem is said to be univariate, whereas for the case that $o > 1$ it is said to be multi-variate. As in the case of classification, let us restrict ourselves to the set of linear hypotheses such that $t(x) = Wx$, with $W : \mathcal{X} \to \mathcal{Y}$ and $W \in \mathbb{R}^{o \times d}$. In the noise-free case, the best hypothesis in H ideally fulfills the interpolation condition $t(x_i) = y_i,\ 1 \leq i \leq N$ for all points in the training set D_N. However, in working with data obtained under real world conditions one will rarely succeed in finding a perfect interpolant in the hypothesis set. Therefore a loss function $l(x, y, \hat{y})$ has to be considered which has to increase the more $\hat{y}$ deviates from y. The following table shows some common used loss functions in the literature:

Loss function	Name
$\|\|y - t(x)\|\|_1$	Least Absolute Deviation
$\|\|y - t(x)\|\|_2^2$	Squared Error
$\max(\|\|y - t(x)\|\|_1 - \epsilon, 0)$	ϵ-insensitive

The most widely used loss function in practice is the squared error. Its popularity can be explained by the following two reasons:

The assumed noise model We assume that the observed data D_N is generated by a true linear relationship among the input and the output but that we observe corrupted outputs. A reasonable assumption is to assume that the corruption is in form of an additive noise term denoted by ξ which is purely random and can be modelled as a zero mean Gaussian variable. For simplicity let us assume in the following that this Gaussian is isotropic. Furthermore we assume that the error is independent to the input. Thus, the observation equation takes the form

$$y_i = Wx_i + \xi, \quad \xi \sim \mathcal{N}(0, \sigma^2).$$

Given the data D_N the corresponding likelihood function of the model W is given by

$$L(W|D_N, \sigma) \propto \prod_{i=1}^{N} \exp^{-\frac{1}{2\sigma^2}\left((y_i - Wx_i)^\top (y_i - Wx_i)\right)}.$$

Thus, the hypothesis which yields the highest likelihood under the assumption that the error is Gaussian can be obtained by $W = \arg\max_W L(W|D_N, \sigma)$. Taking the logarithm of the likelihood yields the identities

$$\begin{aligned} W &= \arg\max_W \log L(W|D_N) = \arg\max_W -\sum_{i=1}^{N} (y_i - Wx_i)^\top (y_i - Wx_i) \\ &= \arg\min_W \sum_{i=1}^{N} ||y_i - t(x_i)||_2^2 \end{aligned}$$

This shows that, the squared error is the natural loss function to take under the assumption that the noise is additive and Gaussian.

Simplicity: Consider the following identity for the summed squared error of all elements in D_N:

$$\sum_{i=1}^{N} ||y_i - Wx_i||^2 = \text{tr}(Y - WX)^\top (Y - WX),$$

where we have arranged all input and output observations as matrices $X = [x_1, \ldots, x_N]$ and $Y = [y_1, \ldots, y_N]$. If we take now the derivative of the right side w.r.t to W and identify critical points, we obtain the relationship $WXX^\top = YX^\top$ and thus the direct solution:

$$W = YX^\top (XX^\top)^{-1}. \tag{10}$$

Consequently, if we use ERM with the squared error loss function we can compute the optimum hypothesis analytically. However, as one might guess from the form of the solution already, the inversion of the matrix $XX^\top$ might lead to difficulties. Consider the case that all inputs x_i, x_j live in a subspace $\mathbb{R}^v$ with $v < d$. In this case, the matrix $XX^\top$ will be rank-deficient such that inversion is not possible anymore. The reason is that there exists not a single unique solution. Quite the contrary is the case: A whole space of valid solutions with $W = W_x + W_{x\perp}$ exists and the new task now is to choose one of them. The way out of this dilemma is again to use a sympathy functional $\Omega[W]$ which favors solutions with special properties. The most widely used sympathy functional for linear regression is the minimum norm regularization scheme which leads to the new optimization problem:

$$W \quad = \quad \arg\min_W L(W) := \arg\min_W \text{tr}(Y - WX)^\top (Y - WX) + \lambda ||W||_2^2.$$

By taking the derivative and identifying the critical point

$$\frac{\partial L}{\partial W} = WXX^\top - YX^\top + \lambda W \stackrel{!}{=} 0,$$

we see that the minimum norm regularized least squares problem can also be solved analytically

$$W = YX^\top (XX^\top + \lambda \mathbb{1})^{-1},$$

which is quite convenient. The algorithm which solves the minimum norm regularized least squares problem is called ridge-regression [38] and we show in A1.2 the dual version for the univariate case.

Algorithm A1.2 RIDGE REGRESSION

Given N pairs $\{(x_1, y_1), \ldots, (x_N, y_N)\} \in \{\mathcal{X} \times \mathbb{R}\}^N$, and a regularization coefficient $\lambda \in \mathbb{R}$ the ridge regression algorithm generates a predictor for the problem

$$w = \arg\min_W \sum_{i=1}^{N} (w^\top \phi_k(x_i) - y_i)^2 + \lambda ||w||^2. \tag{11}$$

The solution to the dual problem with $w = \sum_{i=1}^{N} \alpha_i^* x_i$ is analytically given by

$$\alpha^* \quad = \quad (K + \lambda \mathbf{1})^{-1} Y, \tag{12}$$

where $K_{ij} = x_i^\top x_j$ and $Y = [y_1, \ldots, y_N]^\top$.

So far we have constrained ourselves to linear hypotheses. To obtain nonlinear relationships we would need to change our hypothesis set which would imply considering new algorithms as well. However, we can keep the hypothesis set and thus our introduced algorithms and instead nonlinearly *transform* the inputs. For this purpose, we use kernels which we want to introduce in the following section.

1.2 Beyond linear models: Kernels

In this section we introduce the idea of kernel functions which is a fundamental concept used throughout in this thesis. The basic idea is to map the input data mapped into some other space $\mathcal{F} = \Phi(\mathcal{X})$ such that ideally the nonlinear relationship among $\mathcal{X}$ and $\mathcal{Y}$ is now a *linear* relationship between $\mathcal{F}_k$ and $\mathcal{Y}$. As denoted above we will denote this mapping $\phi : \mathcal{X} \to \mathcal{F}_k$. A very intuitive way to think about ϕ is to consider it as a *blind* mapping which does not care about the task and evaluates a huge amount of non-linear functions $e_i : \mathcal{X} \to \mathbb{R},\ 1 \leq i \leq \dim \mathcal{F}_k$. The new vector $\phi(x) = [e_1(x), e_2(x), \dots]$ specifies the *coordinates* of $\phi(x)$ in the so called feature space $\mathcal{F}_k$. Now, we can use any linear algorithm which takes $\phi(x)$ as input and try to learn a relationship between $\mathcal{X}$ and $\mathcal{Y}$ via the transformed inputs $\phi(x)$. Why is this a good idea at all? In general it is not, since we might have to deal with high-dimensional or even infinite dimensional vectors $\phi(x)$ and direct operation on these vectors is not feasible. Let us reconsider algorithms A1.1 and A1.2. Obviously, for prediction we just need the dot product $y = w^\top \phi(x) + b$. What about training? Since we have given the dual formulation, we see that in the algorithms above at no point we would use the vectors $\phi(x)$ explicitly but again only dot products. The question is now, whether we can evaluate dot products for any given mapping ϕ without considering the elements of $\phi(x)$. Unfortunately, the answer is likely to be no since one might think about mappings ϕ that require explicit representation of $\phi(x)$. However, if we can reformulate the question as: Are there any mappings $\phi(x)$ which correspond to a nonlinear transformation where the dot product can be evaluated without considering the elements of $\phi(x)$? Fortunately, we can affirm this question. As we will see in the following, there exist functions $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which correspond to an Euclidean dot product in some feature space $\mathcal{F}_k$ which is, in general, different from the representation spaces $\mathcal{X}$ and $\mathcal{Y}$. Furthermore, in practice we will only consider this kernel function k without worrying about the implicitly used ϕ mapping. To emphasize that the ϕ is implicitly given by the kernel function k we will use k as a subindex and write ϕ_k. Instead of performing the expensive transformation step explicitly only, the kernel is evaluated and thus the feature transformation is performed only implicitly. Let us now give a formal definition of a kernel function.

Definition D1.1 Kernel Function [78]: *Let Z be a set. A symmetric function $k : Z \times Z \to \mathbb{R}$ is a kernel on Z if the induced quadratic $K_{ij} = k(x_i, x_j)$ is positive definite, thus if*

$$\sum_{i,j}^{n} \alpha_i \alpha_j k(x_i, x_j) = \boldsymbol{\alpha}^\top K \boldsymbol{\alpha}$$

is greater equal zero independent of the choice of $[\alpha_1, \dots, \alpha_n]$ and $\{x_1, \dots, x_n\} \in Z$.

Thus, whenever we have a function with the properties in D1.1 we know that it is the dot product according to some possibly nonlinear mapping of the data. The image $\mathcal{F}_k$ of the implicitly defined feature map $\phi_k : \mathcal{X} \to \mathcal{F}_k$ is called a reproducing kernel Hilbert space. For a functional analysis point of view of kernel functions see [78].

In this thesis, we will also consider data which consists of structured objects such as

graphs or sequences. For these data types it is natural to construct a combined similarity measure from other similarity measures considering partial aspects of the objects only. For example a similarity measure between face images could be combined from similarity measures between eye colors and lip contours. In such situations we will use the following very convenient properties of kernel functions:

Lemma L1.1 Closure [78]:
Kernels are closed under addition and multiplication. Thus, given that $k_1, \ldots, k_l$ are kernel functions it follows that k is a kernel function provided that

a) $k(x, x') = \sum_{i=1}^{l} \alpha_i k_i(x, x') \quad \forall i : \alpha_i > 0,$

b) $k(x, x') = \prod_{i=1}^{l} k_i(x, x').$

c) $k(x, x') = f(x)f(x')$ *and* $f : \mathcal{X} \to \mathbb{R}$ *arbitrary.*

Now given that we have introduced the kernel function, let us give in table 1.1 some examples for kernel functions with their implicitly used feature map $\phi_k(x)$(see [78, 85]).

The kernel function *encapsulates* the data from the algorithm and thus allows us to use the same algorithm on different data types without changing any line in the implementation. Thus, whenever an algorithm can be expressed in kernels as for examples algorithms A1.1 and A1.2 which uses the standard linear dot product (which is a kernel), it can be extended to the nonlinear case by exchanging the kernel function. Finally, let us state the following theorem by [50] which shows another strength of kernel methods.

Theorem T1.3 Representer Theorem [50]
Given some data points $D_N = \{x_i, y_i\}_{i=1}^N$ and under the constraint that $f \in \mathcal{F}_k$, the solution w^* of the variational problem

$$w^* = \arg\min_f \sum_{i=1}^{N} \ell(y_i, x_i, w^\top \phi_k(x_i)) + \Omega[w] \tag{13}$$

has the form:

$$w^* = \sum_{i=1}^{N} \alpha_i \phi_k(x_i). \tag{14}$$

Theorem T1.3 states that whenever we are looking for the optimal linear hypothesis and our hypothesis is element of a reproducing kernel Hilbert space $\mathcal{F}_k$, it is sufficient to look for the expansion coefficients $\boldsymbol{\alpha} \in \mathbb{R}^N$. Thus we are ensured that the best hypothesis is in the span of the mapped data points $\{\phi_k(x_1), \ldots, \phi_k(x_N)\}$. Unfortunately, for practical applications this will be sometimes a disadvantage. Consider the case that we are given

Kernel Function	Implicitly used feature map
$x_1^\top x_2$	id: $\mathcal{X} \to \mathcal{X}$
$(x_1^\top x_2)^d$	$\phi_k(x) : \mathcal{X} \to$ All monomials of order d
$\exp^{\gamma(x_1^\top x_2)}$	$\phi_k(x) : \mathcal{X} \to$ All monomials

Table 1.1: Some kernel functions and their implied feature maps.

a large amount of data points and we would like to obtain a predictor used in a real-time robot control setting. Whenever we want to perform a prediction ${w^*}^\top \phi_k(x^*)$ for a new point x^*, we might be forced to evaluate a kernel function for the whole expansion in (14). Clearly this is not always desirable, and we might need mechanisms to *post-process* the finally obtained hypothesis such that prediction can be performed with less kernel evaluations. The fewer points we are required to keep the fewer kernel functions we need to evaluate. This desirable property is called *sparsity*. We explore in chapter 4 post-processing techniques to obtain sparse expansions.

Sometimes, the operation of an algorithm can not be expressed in dot-products only and thus we can not use kernels directly. For example, in chapter 2 we introduce a linear regression algorithm which uses a sympathy functional leading to operations that can not be expressed with dot-products only. In this situation, we still would like to use this linear algorithm for non-linear scenarios and avoid accessing entries of $\phi_k(x)$. Fortunately, we can use the fact that given N points as training data the optimal hypothesis must be in the span of these N training data. Thus instead of considering all coordinates of a vector $\phi_k(x)$, it is sufficient to project $\phi_k(x)$ on the N-dimensional subspace spanned by the training points and to consider coordinates in this N-dimensional subspace only. We thus need to construct an orthogonal basis $\boldsymbol{V} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_N]$ such that $\boldsymbol{V}$ spans the same space as the training set. Given such a basis we can then project a new mapped point $\phi_k(x)$ onto this basis by $\boldsymbol{V}^\top \phi_k(x) = \left[\boldsymbol{v}_1^\top \phi_k(x), \ldots, \boldsymbol{v}_N^\top \phi_k(x)\right]$. Note, that the projection requires the calculation of dot products only. Clearly, all $\boldsymbol{v}_i$ are in the span of the training data, thus we are ensured that there exists coefficients $\boldsymbol{\alpha}^j \in \mathbb{R}^N$ such that $v_j = \sum_{i=1} \alpha_i^j \phi_k(x_i)$. The question is now which orthogonal basis to choose, since there are infinitely many possible ones. A classic (linear) algorithm is principal component analysis (PCA), trying to align the basis elements $\boldsymbol{v}_i$ such that the empirical variance along $\boldsymbol{v}_i$ is larger than $\boldsymbol{v}_j$ for $i < j$. The basis elements are called principal directions, the extracted coordinates are called principal values. For a discussion on PCA see [44]. In A1.3 we show the kernelized version of PCA due to [80] which we abbreviate as kPCA.

Now that we have introduced kernel functions and their use in supervised learning algorithms we are ready to explain the method of Kernel Dependency Estimation.

1.3 KDE

Learning general functional dependencies between arbitrary input and output spaces is one of the main goals in machine learning. As we saw, classical supervised learning frameworks deal with the case that the outputs are $\mathcal{Y} \equiv \{0, 1\}$ for classification or $\mathcal{Y} \equiv \mathbb{R}^o$ for regression. As we have discussed above kernel functions on the input side encapsulate the type of the input data from the algorithm. Thus the remaining difference between

---

Algorithm A1.3 Kernel Principal Component Analysis

Given N patterns $\{x_1, \ldots, x_N\} \in \mathcal{X}^N$ and a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ the kernel PCA algorithm generates N feature extractors $v_j = \frac{1}{\sqrt{\lambda_j}} \sum_{i=1}^N \alpha_i^j \phi_k(x_i)$ with $j = 1, \ldots, N$ and $\alpha^j \in \mathbb{R}^N$ obtained by the eigenvalue problem:

$$K\alpha^j = \frac{\lambda_j}{N} \alpha^j \tag{15}$$

where $K_{ij} = k(x_i, x_j)$.

---

regression and classification are due to the different output type. Indeed, if $\mathcal{Y}$ is not a subset of $\mathbb{R}$ but a general set, it cannot be easily modelled well in the standard settings of regression or classification. In this sense, the output data type dictates the choice of the learning algorithm. [5] For example, if the target is a real vector the algorithm might minimize the squared distance between prediction and target, whereas if the target is a string, the algorithm might minimize a distance measure appropriate for strings (for example the Hamming distance or the edit distance [85]). *Classically, the implementation of a supervised learning procedure depends on the nature of the output.* The idea of Kernel Dependency Estimation [99] is to use the kernel idea not only to encapsulate the inputs but also the outputs. Thus the same algorithm could be used independently of the input and of the output data-type. Let us look at the KDE algorithm in more detail.

1.3.1 A rough sketch: Divide and Conquer

Assuming that the loss function $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ depends only on the output, it can be interpreted as a *metric* measuring the distance between targets and predictions in the output space. This alternative interpretation of the loss function allows us to think of a hypothetical space $\mathcal{F}_l$ associated to the loss function l with the following properties:

a) Elements $\boldsymbol{y}$ of $\mathcal{F}_l$ are obtained by a possibly nonlinear map $\phi_l : \mathcal{Y} \to \mathcal{F}_l$.

b) The Euclidean distance among two points $\phi_l(y)$ and $\phi_l(\hat{y})$ in the space $\mathcal{F}_l$ equals to the loss $l(y, \hat{y})$, i.e.:

$$||\phi_l(y) - \phi_l(\hat{y})||^2_{\mathcal{F}_l} = l(y, \hat{y}).$$

If such a space $\mathcal{F}_l$ exists, one could use a learning algorithm that selects the mapping $t : \mathcal{X} \to \mathcal{Y}$ by minimizing the squared loss in $\mathcal{F}_l$ instead of the problem or data-type-dependent loss function l. Thus, with $t(x_i)$ being our predictions, we obtain the equivalence

$$t^* = \arg\min_t \sum_{i=1}^{N} l(y_i, t(x_i)), \tag{16}$$

$$\Leftrightarrow \quad \arg\min_t \sum_{i=1}^{N} ||\phi_l(y_i) - \phi_l(t(x_i))||^2. \tag{17}$$

What is the advantage of doing so? As we saw above, the squared error is used in a regression setting. Thus right now, provided that we have a map ϕ_l we turned our supervised learning problem into a squared-error regression problem. From now on, as for the input case, we will only consider those ϕ_l which are implicitly given by some kernel function implying that $\mathcal{F}_l$ is a reproducing kernel Hilbert space. Using now a kernel function for the output data that encodes information about the used loss function, we have encapsulated all information regarding the outputs in the kernel function. Let us denote such an *output* kernel function by k_y and rewrite (17) as follows:

$$\begin{aligned} t^* &= \arg\min_t \sum_{i=1}^{N} ||\phi_l(y_i) - \phi_l(t(x_i))||^2 \\ &= \arg\min_t \sum_{i=1}^{N} k_y(y_i, y_i) + k_y(t(x_i), t(x_i)) - 2k_y(t(x_i), y_i). \end{aligned} \tag{18}$$

[5] Note, that in the past this was a quite basic property of the choice of computer algorithms in general. The implementation was coupled to the processed data types. The principles of programming changed with the appearance of the object orientation[88] and the design pattern [26] frameworks.

As one can see from (18) this optimization problem is likely to be hard to solve. However, we can split this optimization problem into parts by dividing the mapping t into:

$$t = \Gamma \circ T \circ \phi_k, \tag{19}$$

where

- ϕ_k is the feature map, embedding elements from input space into the feature space $\mathcal{F}_k$. This encapsulates the input data.
- $T : \mathcal{F}_k \to \mathcal{F}_l$ is the unknown map operating between feature spaces, which estimates a point $\boldsymbol{y} \in \mathcal{F}_l$ in the output feature space given a point $\boldsymbol{x} = \phi_k(x)$ in the input feature space.
- $\Gamma : \mathcal{F}_l \to \mathcal{Y}$ is the *pre-image* map which maps elements of $\mathcal{F}_l$ back to the original output space $\mathcal{Y}$. (Ideally it is the inversion of ϕ_l, i.e. $\Gamma = \phi_l^{-1}$, see below)

Now, we have obtained a *data-type independent* regression problem: Choosing the right map T among the feature spaces. Once we provide mappings ϕ_l, ϕ_k and the pre-image map Γ we solve the inference problem in feature space. Note that the maps ϕ_l, Γ are independent from the input and likewise ϕ_k depends only on the input. Therefore, choosing these maps does not require to solve the original learning task and therefore is easier than the original problem at the beginning. Furthermore, although it looks difficult to choose the operator T, note that in principle it is a standard multivariate regression problem. To see this, given any finite data set D_N, we can apply kPCA (see A1.3) to the input and output data independently and retrieve the orthogonal bases $\boldsymbol{W}$ and $\boldsymbol{V}$. Once we have obtained these bases we can extract input and output coordinates of $\phi_k(x_i)$ respective $\boldsymbol{W}$ and $\phi_l(y_i)$ respective $\boldsymbol{V}$ yielding the new coordinate representation of the training data: $\hat{D}_N = \{\hat{x}_i, \hat{y}_i\}_{i=1}^N$ with $\hat{x}_i, \hat{y}_i \in \mathbb{R}^N$. Using the coordinate representation we can then use for example a standard regression algorithm like ridge regression (see A1.2) to obtain the mapping $\hat{T} : \mathbb{R}^N \to \mathbb{R}^N$. Thus our final operator is given by the concatenation $T = \boldsymbol{V} \circ \hat{T} \circ \boldsymbol{W}$. To make the difference between the map T and $\hat{T}$ more explicit we denote the map between feature spaces by $T_{\mathcal{F}} : \mathcal{F}_k \to \mathcal{F}_l$ and as a counterpart the original map as $t_{\mathcal{Z}} : \mathcal{X} \to \mathcal{Y}$, with $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$.

Note that given the same input and output kernels, different coordinate representation of the input and output data will give rise to different regression problems. For example if the data is collinear in feature space, one can try to restrict the dimensionality first. This is the basic motivation for regression algorithms we explore in chapter 2.

Once we have obtained our map $T_{\mathcal{F}}$ we can use it to predict the output coordinates of a new presented input point $\phi_k(x^*)$. Now, since at the end of the day we have to predict elements from $\mathcal{Y}$, we need to generate an element from $\mathcal{Y}$ such that ideally

$$T\phi_k(x^*) = \phi_l(y^*).$$

This is the job of the map $\Gamma : \mathcal{F}_l \to \mathcal{Y}$ which ideally is the inverse of the output feature map ϕ_l and therefore is as well problem-dependent. [6] Unfortunately, it will often turn out that Γ will not be the exact inverse of ϕ_l since the inverse will not exist and the point $T\phi_k(x^*)$ does not need the have a valid *pre-image*. We will investigate in chapter 3 techniques and possible strategies to solve pre-image problems. Finally, let us summarize all decompositions and maps in KDE in figure 1.2.

[6] Note, however that it is independent of the input.

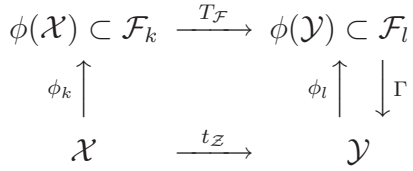


Figure 1.2: Mappings between original sets $\mathcal{X}, \mathcal{Y}$ and corresponding feature spaces $\mathcal{F}_k$, $\mathcal{F}_l$ in KDE.

1.4 Structure of this thesis

The first part of this thesis will focus on various algorithmic extensions to this algorithm not focusing on the imitation learning task but keeping the scope very general allowing the techniques to be used in other fields as well. In particular we will introduce in chapter 2 a regression method which aims at identifying relevant subspaces in high dimensional input and output feature spaces relevant for prediction during the learning stage. In chapter 3 we discuss algorithms for the pre-image problem which is the most critical issue in the usage of KDE. In addition man-machine-interaction requires following human actions in realtime. Since in general kernel methods tend to be too slow for most applications in robotics, in chapter 4 we investigate algorithms which can be used for post-processing models obtained from kernel algorithms to increase the testing speed. In the last part of this thesis we apply the method of KDE to the task of imitating pose. We start in chapter 5 with an approach based on perfect environmental knowledge which allows us to formulate the imitation problem as an optimization problem. Afterwards in chapter 6 we investigate a different approach based on the KDE algorithm. Figure 1.3 can be used as reading guide.

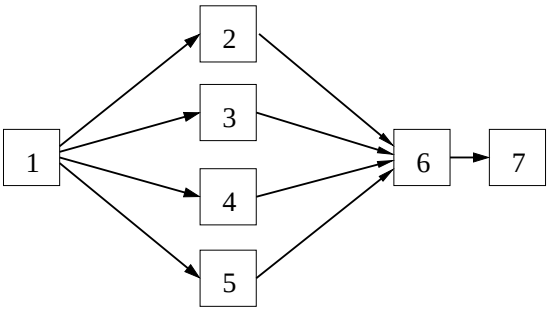


Figure 1.3: Chapter dependency.

Part I

Kernel Dependecy Estimation

Chapter 2

Multivariate regression via Stiefel constraints.

Wissen und Können ist wie Blüte und Frucht.
– Holger Weiss –

One important subproblem in the framework of Kernel Dependency Estimation(KDE) is the problem of multivariate regression, that is to estimate from a given coordinate input vector $x \in \mathbb{R}^N$ the coordinates of an output vector $y \in \mathbb{R}^N$. This chapter introduces a novel multivariate regression technique based on regularization via rank constraints, which is particular suited to problems with high-dimensional outputs.

2.1 Introduction

The problem of regressing between a high-dimensional input space and a *continuous, univariate* output has been studied in considerable detail: classical methods are described in [35], and methods applicable when the input is in a reproducing kernel Hilbert space are discussed in [78]. When the output dimension is high (or even infinite), however, it becomes inefficient or impractical to apply univariate methods separately to each of the outputs, and specialized multivariate techniques must be used.

In this chapter we propose a novel method for regression between two spaces $\mathbb{R}^d$ and $\mathbb{R}^o$, where both spaces can have arbitrarily large dimension. Our algorithm works by choosing low-dimensional subspaces in both $\mathbb{R}^d$ and $\mathbb{R}^o$ for each new set of observations made, and finding the mapping between these subspaces for which a particular loss is small.[1] There are several reasons for learning a mapping between low-dimensional subspaces, rather than between $\mathbb{R}^d$ and $\mathbb{R}^o$ in their entirety. First, $\mathbb{R}^d$ and $\mathbb{R}^o$ may have high dimension, yet our data are generally confined to smaller subspaces. Second, the outputs may be statistically dependent, and learning all of them at once allows us to exploit this dependence. Third, it is common practice to ignore certain directions in the input and/or output spaces, which decreases the variance in the regression coefficients (at the expense of additional bias): this is a form of regularization.

Given a particular subspace dimension, classical multivariate regression methods use a variety of heuristics for subspace choice.[2] The mapping between subspaces is then

[1] The loss is specified by the user.

[2] For instance, Principal Component Regression generally retains the input directions with highest *variance*, whereas partial least squares (PLS) approximates the input directions along which *covariance*

achieved as a second, independent step. These methods generally represent a compromise between generalization performance and computational simplicity. By contrast, our method, Multivariate Regression with Stiefel Constraints (MRS), jointly optimizes over the subspaces *and* the mapping; its goal is to find the subspace/mapping *combination* with the smallest possible loss. Drawing on results from differential geometry (see [23, 42]), we represent each subspace projection operator as an element of the Stiefel manifold (see [23]). Our method then conducts gradient descent over these projections. On moderately sized datasets, the subspace/mapping estimation problem may be solved using a batch method, in which optimization is carried out over the entire data set. When the data set is large, however, this can have prohibitive computational and memory requirements. For such cases, we propose an on-line variant of our algorithm, called Sequential Multivariate Regression with Stiefel Constraints (S-MRS), which processes a series of small subsets of the data, rather than requiring all of it at once. The sequential approach has two advantages: it allows us to easily update our prediction in the light of new observations, and to learn from large data sets by breaking the problem down into smaller learning tasks. The choice of subspaces and mapping must then be constrained, in that both are not permitted to change too much from those previously learned. This ensures that we do not disregard past observations when new data comes in. The similarity of the new solution to the prior solutions is enforced using a *regularizing* term, [3] which is added to the original batch-based loss. A disadvantage of all optimization-based approaches is that the rank constraint requires the optimization to be performed over a non-convex set. To overcome the problems caused by non-convexity (e.g. local minima), we investigate in section 2.5 a method which aims at finding the maximum a posteriori solution for a probabilistic model using a sampling technique.

We begin our discussion in section 2.2 with some basic definitions, and give a formal description of the multivariate regression setting (both batch and sequential). We then summarize various classical methods for multivariate regression in section 2.3. In each case, the relevant subspaces in $\mathbb{R}^d$ and $\mathbb{R}^o$ are described, and the heuristics used to determine these subspaces are elucidated. Next, in section 2.4, we introduce the MRS procedure for differentiable loss functions, in particular the ℓ_2 losses, and the S-MRS procedure for the ℓ_2 loss. In the case that the loss function is not differentiable we propose to use a maximum likelihood predictor obtained by utilizing Bayesian inference via a Markov Chain Monte Carlo approach. Thus in section 2.5 we introduce a probabilistic approach for MRS and investigate the application of different loss functions.

2.2 Problem setting and motivation

We first describe our regression setting in more detail, and introduce the variables we will use. We are given m pairs of input and output variables, $D_N := ((\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_m, \mathbf{y}_m))$, where $\mathbf{x}_i \in \mathcal{F}_x$, $\mathbf{y}_i \in \mathcal{F}_y$, and $\mathcal{F}_x$ and $\mathcal{F}_y$ are reproducing kernel Hilbert spaces with dimensions l_x and l_y respectively. We write the matrices of the *centered* observations as

$$\mathbf{X} := \left[\begin{array}{ccc} \mathbf{x}_1 & \ldots & \mathbf{x}_m \end{array} \right] \mathbf{H}, \quad \mathbf{Y} := \left[\begin{array}{ccc} \mathbf{y}_1 & \ldots & \mathbf{y}_m \end{array} \right] \mathbf{H},$$

where $\mathbf{H} := \mathbf{I} - \frac{1}{m}\mathbf{1}\mathbf{1}^\top$, and $\mathbf{1}$ is the $m \times 1$ matrix of ones.

with the outputs is high.

[3] This regularization should not be confused with the regularizer used in ridge regression, which can *also* be used to constrain the solution obtained for each data *subset*; or indeed for the entire data set in the batch case.

We now specify our learning problem: given observations $\mathbf{X}$ and $\mathbf{Y}$, a loss function $\mathsf{L}(\mathbf{Y}, \mathbf{X}, \mathbf{F}_{(r)})$, and a regularizer Ω_d (this can be for instance the 2-norm regularization as in ridge regression: see section 2.4.4), we want to find the best predictor $\mathbf{F}_{(r)}$, defined as

$$\mathbf{F}_{(r)} \quad = \quad \min_{\mathbf{G} \in \mathcal{H}_{(r)}} \mathsf{L}(\mathbf{Y}, \mathbf{X}, \mathbf{G}) + \Omega_d(\mathbf{G}), \tag{1}$$

where $\mathcal{H}_{(r)}$ is the set of all rank-constrained linear mappings

$$\mathcal{H}_{(r)} := \left\{ \mathbf{F} : \mathbb{R}^d \to \mathbb{R}^o \,|\, \mathbf{rank}\ \mathbf{F} = r \right\} \tag{2}$$

from the input space $\mathbb{R}^d$ to the output space $\mathbb{R}^o$. This rank constraint is crucial to our approach: it allows us to restrict ourselves to subspaces *smaller* than those spanned by the input and/or output observations, which can reduce the variance in our estimate of the mapping $\mathbf{F}_{(r)}$ while increasing the bias. We select the rank that optimizes over this bias/variance trade-off using cross validation.

We now transform the rank constraint in (1) and (2) into a form more amenable to optimization. By diagonalizing the predictor $\mathbf{F}_{(r)}$ via its singular basis, we obtain

$$\mathbf{F}_{(r)} \quad = \quad \mathbf{V}_{(r)} S_{(r)} {\mathbf{W}_{(r)}}^\top, \tag{3}$$

where

$$\begin{aligned} {\mathbf{V}_{(r)}}^\top \mathbf{V}_{(r)} &= \mathbb{1}_{r,r}, & (4)\\ {\mathbf{W}_{(r)}}^\top \mathbf{W}_{(r)} &= \mathbb{1}_{r,r}, & (5)\\ S &\in \text{diagonal } \Re^{r \times r}. & (6) \end{aligned}$$

In other words, $\mathbf{W}_{(r)} \in \mathbb{S}(l_y, r)$ and $\mathbf{V}_{(r)} \in \mathbb{S}(l_x, r)$, where $\mathbb{S}(n, r)$ is called the *Stiefel* manifold, and comprises the matrices with n rows and r orthonormal columns. In the case of the squared loss $\mathsf{L}_2(\mathbf{Y}, \mathbf{X}, \mathbf{F}_{(r)}) = ||\mathbf{Y} - \mathbf{F}_{(\mathbf{r})}\mathbf{X}||^{\mathbf{2}}_{\mathbf{F}}$, finding a rank constrained predictor (3) is thus equivalent to finding the triplet $\theta = (\mathbf{V}_{(r)}, S_{(r)}, \mathbf{W}_{(r)})$ for which

$$\theta \quad = \quad \underset{\mathbf{V}_{(r)}, S_{(r)}, \mathbf{W}_{(r)}}{\arg\min} \|\mathbf{Y} - \mathbf{V}_{(r)} S_{(r)} {\mathbf{W}_{(r)}}^\top \mathbf{X}\|_F^2, \tag{7}$$

subject to constraints (4)-(6) [4]. We will refer to $\mathbf{W}_{(r)}$ and $\mathbf{V}_{(r)}$ as *feature matrices*.

It is clear from (7) that $\mathbf{W}_{(r)}$ and $\mathbf{V}_{(r)}$ determine particular subspaces in $\mathbb{R}^d$ and $\mathbb{R}^o$ respectively, and that the regression procedure is a mapping between these subspaces. A number of classical multivariate regression methods also have this property, although the associated subspaces are not determined according to the criteria we propose or are only applicable for a particular loss function. In section 2.3, we will review the subspace selection methods used in three established regression algorithms, before returning in section 2.4 to the solution of (7). First, however, we describe the on-line framework used in S-MRS.

2.2.1 On-line setting

In the sequential case, we do not observe the examples all at once: rather, at every time instance k, we obtain only m observations. These are written as

$$\mathbf{X_k} := \left[\ \mathbf{x}_{k_1} \quad \dots \quad \mathbf{x}_{k_m}\ \right], \quad \mathbf{Y_k} := \left[\ \mathbf{y}_{k_1} \quad \dots \quad \mathbf{y}_{k_m}\ \right], \quad \text{and } \boldsymbol{Z}_k = [\boldsymbol{X}_k, \boldsymbol{Y}_k].$$

[4] This is a more general form of the *Procrustes* problem, see [6], for which $\mathbf{F}_{(r)}$ is orthogonal rather than being rank constrained.

The on-line setting requires (1) and (2) to be slightly modified: thus, for a particular $\mathbf{G} \in \mathcal{H}_{(r)}$, we can define a loss $\mathsf{L}(\boldsymbol{Z}_k, \mathbf{G})$ measuring the prediction error of $\mathbf{G}$ on $\boldsymbol{Z}_k$, and a regularization functional $\Omega_o(\mathbf{G}, \mathbf{F}_{(r)}^{(k-1)})$ that penalizes large differences between $\mathbf{G}$ and a reference mapping $\mathbf{F}_{(r)}^{(k-1)}$. We then want to find the best predictor $\mathbf{F}_{(r)}^{(k)}$, defined as

$$\mathbf{F}_{(r)}^{(k)} = \underset{\mathbf{G}\in\mathcal{H}_{(r)}}{\arg\min}\, \mathcal{L}(\mathbf{Z_k}, \mathbf{G}, \mathbf{F}_{(r)}^{(k-1)}) \tag{8}$$

$$= \underset{\mathbf{G}\in\mathcal{H}_{(r)}}{\arg\min}\, \mathsf{L}(\mathbf{Z}_k, \mathbf{G}) + \Omega_o(\mathbf{G}, \mathbf{F}_{(\mathbf{r})}^{(\mathbf{k-1})}). \tag{9}$$

We may understand the purpose of Ω_o by reference to the (non-sequential) ridge regression method, for which $\mathbf{F}_{(r)}^{(k-1)} = \mathbf{0}$ and

$$\Omega_o(\mathbf{G}, \mathbf{0}) = \|\mathbf{G}\|_F^2 :$$

in this case the regularization prevents the norm $\|\mathbf{G}\|_F^2$ from growing too large. In the on-line case, the reference mapping is the solution $\mathbf{F}_{(r)}^{(k-1)}$ obtained previously, in keeping with our goal of choosing a new mapping close to the former solution. By way of comparison, the Kalman-Bucy filter can also be written in the form (8) with an analogous reference mapping, see [**?**]. The specific form taken by Ω_o in our multivariate regression setting will be described in section 2.4.5.

We again transform the rank constraint in (8) using the decomposition in (3) and (4)-(6). Thus, for the kth set of observations, finding a rank constrained predictor (3) is equivalent to finding the triplet $\theta = (\mathbf{V}_{(r)}, S_{(r)}, \mathbf{W}_{(r)})$ for which

$$\theta = \underset{\mathbf{V}_{(r)}, S_{(r)}, \mathbf{W}_{(r)}}{\arg\min}\, \mathcal{L}(\mathbf{V}_{(r)} S_{(r)} \mathbf{W}_{(r)}{}^{\top}, \mathbf{F}_{(r)}^{(k-1)}) \tag{10}$$

subject to constraints (4)-(6).

2.3 Existing Multivariate Techniques

In this section, we review existing techniques for reduced rank regression. In particular we consider reduced rank regression (RRR), principal component regression (PCR) and partial least squares (PLS), paying particular attention to the way in which a small number r of features is chosen. We will discuss the different construction of the mappings $\mathbf{F}_{(r)}$ (the features may be chosen in the input *or* output spaces, depending on the algorithm; the rank of the mapping is unaffected). We consider only the batch case for each algorithm. Since each of these methods rely on the L_2 loss

$$\mathsf{L}_2(\mathbf{Y}, \mathbf{X}, \mathbf{F}_{(r)}) = \|\mathbf{Y} - \mathbf{F}_{(r)}\mathbf{X}\|_F^2, \tag{11}$$

we begin with a description of the *least squares solution* associated with this loss, and demonstrate how this solution changes when the *inputs* are projected onto a small number of features. This solution is then used as a basis for principal component regression (PCR) and partial least squares (PLS), although the choice of features differs between these algorithms. Finally, we describe the reduced rank solution where the features are obtained by solving an eigenvalue problem.

2.3.1 Restricted Least Squares Regression

The multivariate least squares solution is equivalent to l_y separate univariate problems; thus, we first describe the univariate case. We wish to solve

$$\mathbf{b}^*_{(r)} \quad := \quad \arg\min_{\mathbf{b}} \left\| \mathbf{y} - \mathbf{X}^\top \mathbf{W}_{(r)} \mathbf{b} \right\|^2, \tag{12}$$

where $\mathbf{y}^\top = \left[\begin{array}{ccc} y_1 & \dots & y_m \end{array} \right] \mathbf{H}$, and the solution ${\mathbf{f}_{(r)}}^* := \mathbf{W}_{(r)} \mathbf{b}^*_{(r)}$ is expressed in terms of a linear combination of the columns of $\mathbf{W}_{(r)}$ (*i.e.*, the features) with coefficients $\mathbf{b}$. Taking the derivative with respect to $\mathbf{b}$ and setting this to zero yields

$$\mathbf{b}^*_{(r)} = \left({\mathbf{W}_{(r)}}^\top \mathbf{X} \mathbf{X}^\top \mathbf{W}_{(r)} \right)^{-1} {\mathbf{W}_{(r)}}^\top \mathbf{X} \mathbf{y},$$

and thus

$${\mathbf{f}_{(r)}}^* \;=\; \mathbf{W}_{(r)} \left({\mathbf{W}_{(r)}}^\top \mathbf{X} \mathbf{X}^\top \mathbf{W}_{(r)} \right)^{-1} {\mathbf{W}_{(r)}}^\top \mathbf{X} \mathbf{y}. \tag{13}$$

It helps in interpreting the features if the columns of $\mathbf{W}_{(r)}$ are orthogonal or conjugate, but this is certainly not required. In particular, all that matters is the *subspace* spanned by the columns of $\mathbf{W}_{(r)}$, in that we can replace in equation (13) $\mathbf{W}_{(r)}$ by $\mathbf{U}_{(r)} := \mathbf{W}_{(r)} \mathbf{C}$ for any invertible matrix $\mathbf{C}$ and still get the same ${\mathbf{f}_{(r)}}^*$.

We now describe the multivariate case. In the absence of any restriction of the input to a subspace, we would solve

$$\mathbf{F}^* \quad := \quad \arg\min_{\mathbf{F}} \|\mathbf{Y} - \mathbf{F}\mathbf{X}\|_F^2 \tag{14}$$

$$\quad = \quad \sum_{k=1}^{l_y} \left\| \mathbf{y}_k^\top - \mathbf{f}_k^\top \mathbf{X} \right\|^2 \tag{15}$$

where $\mathbf{y}_k^\top$ and $\mathbf{f}_k^\top$ denote the kth *row* of $\mathbf{Y}$ and $\mathbf{F}$ respectively, $||\cdot||_F$ denotes the Frobenius norm and $\mathbf{F}$ is $l_y \times l_x$. An important point to note in (15) is that each coordinate in the output $\mathbf{y}$ is predicted entirely using a particular row in $\mathbf{F}$. A least squares solution is thus

$$(\mathbf{F}^*)^\top := \left(\mathbf{X}\mathbf{X}^\top \right)^\dagger \mathbf{X}\mathbf{Y}^\top, \tag{16}$$

which is the concatenation of the separate least squares solutions for the l_y individual rows of $\mathbf{Y}$. We can again project the input $\mathbf{X}$ onto $\mathbf{W}_{(r)}$; the least squares solution is then

$$\left({\mathbf{F}_{(r)}}^* \right)^\top := \mathbf{W}_{(r)} \left({\mathbf{W}_{(r)}}^\top \mathbf{X} \mathbf{X}^\top \mathbf{W}_{(r)} \right)^{-1} {\mathbf{W}_{(r)}}^\top \mathbf{X} \mathbf{Y}^\top, \tag{17}$$

which has rank at most r.

2.3.2 Principal Component Regression

The simplest method of selecting features in the input space is principal component regression. We write the singular value decomposition of $\mathbf{X}$ as

$$\mathbf{X} = \left[\begin{array}{cc} \mathbf{W}_{(r)} & \mathbf{W}_{(r)\perp} \end{array} \right] \left[\begin{array}{cc} \mathbf{S}_{(r)} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_{(r)\perp} \end{array} \right] \left[\begin{array}{c} {\mathbf{U}_{(r)}}^\top \\ {\mathbf{U}_{(r)\perp}}^\top \end{array} \right],$$

where the r largest singular values are in $\mathbf{S}_{(r)}$, and the $l_x - r$ smallest in $\mathbf{S}_{(r)\perp}$. We can approximate $\mathbf{X}$ using only

$$\widehat{\mathbf{X}} := \mathbf{W}_{(r)} \mathbf{S}_{(r)} {\mathbf{U}_{(r)}}^\top,$$

where we retain the directions of $\mathbf{X}$ with highest variance (which are assumed to have greatest significance in predicting $\mathbf{y}$). The features $\mathbf{W}_{(r)}$ are then used in (17).

2.3.3 Partial Least Squares in one and more dimensions

The method of partial least squares introduced by [101] is widely applied in chemometrics, where it is known to perform particularly well in cases where the input data are highly collinear. Our discussion in this section largely follows [20, 37] where the standard NIPALS training algorithm is introduced. A modification in which PLS is performed in the dual was proposed in [55], and the algorithm was *kernelized* in [72] and in [4] for use when the output space is a reproducing kernel Hilbert space.

We again start with the one-dimensional case, since it forms the basis of the multi-dimensional algorithm. We give two descriptions of the feature matrix $\mathbf{W}_{(i)} := \left[\begin{array}{ccc} \mathbf{w}_1 & \dots & \mathbf{w}_i \end{array} \right]$ obtained: the first, which is by far the best known, uses orthogonal features, whereas in the second the features are conjugate with respect to $\mathbf{X}\mathbf{X}^\top$. The subspaces spanned by the features are in both cases identical, however. We initialize with $\mathbf{X}_0 := \mathbf{X}$, and begin at $i = 1$. We then iterate the following steps over i.

$$\begin{aligned} \mathbf{w}_i &= \mathbf{X}_{i-1}\mathbf{y} && (18) \\ \mathbf{t}_i &= \mathbf{X}_{i-1}^\top \mathbf{w}_i && (19) \\ \mathbf{X}_i &= \mathbf{X}_{i-1}\left(\mathbf{I} - \frac{\mathbf{t}_i \mathbf{t}_i^\top}{\mathbf{t}_i^\top \mathbf{t}_i}\right) && (20) \end{aligned}$$

A useful result is the following theorem:

Theorem T2.1

The factors $\mathbf{t}_i$ are mutually orthogonal.

Proof. After $j \geq 1$ steps, $\mathbf{X}_{(i+j-1)} = \mathbf{X}_{(i-1)} \prod_{l=i}^{l=i+j-1} \left(\mathbf{I} - \frac{\mathbf{t}_{(l)}\mathbf{t}_{(l)}^\top}{\mathbf{t}_{(l)}^\top \mathbf{t}_{(l)}}\right)$. Thus the rows of $\mathbf{X}_{(i+j-1)}$ are orthogonal to each of $\left\{\mathbf{t}_{(i)}, \dots, \mathbf{t}_{(i+j-1)}\right\}$ (a special case being $j = 1$, and $\mathbf{X}_{(i)}$ having rows orthogonal to $\mathbf{t}_{(i)}$). Then since $\mathbf{t}_{(i+j)}$ is a linear combination of the rows of $\mathbf{X}_{(i+j-1)}$ (from (19)), it follows that $\mathbf{t}_{(i)}$ is orthogonal to $\mathbf{t}_{(i+j)}$ for all $j \geq 1$. □

Equation (20) can be rewritten more compactly as

$$\mathbf{X}_i = \left[\mathbf{I} - \mathbf{T}_{(i)}\left(\mathbf{T}_{(i)}^\top \mathbf{T}_{(i)}\right)^{-1} \mathbf{T}_{(i)}^\top\right] \mathbf{X}, \tag{21}$$

with $\mathbf{T}_{(i)} := [\mathbf{t}_{(1)} \dots \mathbf{t}_{(i)}]$. It is proved in [37] that following r iterations, PLS yields a predictor as given in (13).

A motivation often employed to justify PLS[5] is that the *covariance* between input and output is used to weight each component of $\mathbf{x}$ when predicting y. This is certainly true for the choice of the first feature $\mathbf{w}_1 = \boldsymbol{X}^\top y$, which is equal to this covariance. This explanation is less useful in subsequent iterations, however, since the features $\mathbf{w}_i$ are *not* chosen by projecting out $\mathbf{W}_{i-1}$ from $\mathbf{X}$, and computing the covariance[6] using this deflated $\mathbf{X}$.

[5] Note that PLS was originally derived in terms of a factor model, in which an underlying variable $\mathbf{t}$ is assumed to generate $\mathbf{x}$ and y. This motivation justifies the deflation procedure, but gives an incomplete view of the predictive performance.

[6] If we did project out the previous $\mathbf{w}$, our update (20) would become $\mathbf{X}_{(i)} = \left(\mathbf{I} - \frac{\mathbf{w}_{(i)}\mathbf{w}_{(i)}^\top}{\mathbf{w}_{(i)}^\top \mathbf{w}_{(i)}}\right)\mathbf{X}_{(i-1)} = \mathbf{X}_{(i-1)}\left(\mathbf{I} - \frac{\mathbf{y}_{(i)}\mathbf{t}_{(i)}^\top}{\mathbf{y}_{(i)}^\top \mathbf{t}_{(i)}}\right)$; thus the intuition that the features maximize the covariance is only true to the extent that $\mathbf{t}$ approximates $\mathbf{y}$. The SIMPLS algorithm, introduced by [18], in fact updates $\mathbf{X}$ by projecting out $\mathbf{w}$, although this algorithm also uses a deflation on $\mathbf{y}$ which causes it to return the same features $\mathbf{w}$ as NIPALS.

In the univariate case, there is a more satisfying explanation for PLS performance than the one given above, which is that PLS yields an identical solution to conjugate gradient (CG) descent on $\left\|\mathbf{y} - \mathbf{X}^\top \mathbf{f}\right\|^2$, with respect to the mapping $\mathbf{f}$. The CG method, [86], builds a feature matrix $\mathbf{D}_{(i)} := [\mathbf{d}_1, \ldots, \mathbf{d}_i]$ by setting each $\mathbf{d}_{i+1}$ as close as possible to the direction of steepest descent at the present solution $\mathbf{f}_{(i)} = \mathbf{D}_{(i)} \left(\mathbf{D}_{(i)}^\top \mathbf{X}\mathbf{X}^\top \mathbf{D}_{(i)}\right)^{-1} \mathbf{D}_{(i)}^\top \mathbf{X}\mathbf{y}$, subject to being conjugate to all the columns in $\mathbf{D}_{(i)}$ with respect to $\mathbf{X}\mathbf{X}^\top$. The solution returned after r steps is then obtained by setting $\mathbf{W}_{(r)} = \mathbf{D}_{(r)}$ in (13). The choice of conjugate features $\mathbf{D}_{(r)}$ helps to explain the good performance of PLS when the inputs are highly collinear[7], and these features serve a clear purpose in minimizing the loss (being approximately aligned with the directions of steepest descent).

The simplest method for generalizing to the multivariate case is to perform univariate regression on each output variable: indeed, this was found to outperform the multivariate NIPALS algorithm in [20]. This approach is obviously not feasible when l_y is very large; thus regression to a lower dimensional subspace in $\mathcal{F}_\mathcal{Y}$ is required. There are several multivariate PLS methods that accomplish this task. We describe NIPALS, which is the most widely used. Features are generated as follows:

$$\mathbf{w}_i = \arg\max_{\|\mathbf{w}\| \leq 1} \mathbf{w}^\top \mathbf{X}_{i-1} \mathbf{Y}^\top \mathbf{Y} \mathbf{X}_{i-1}^\top \mathbf{w} \tag{22}$$

$$\mathbf{t}_i = \mathbf{X}_{i-1}^\top \mathbf{w}_i \tag{23}$$

$$\mathbf{X}_i = \mathbf{X}_{i-1} \left(\mathbf{I} - \frac{\mathbf{t}_i \mathbf{t}_i^\top}{\mathbf{t}_i^\top \mathbf{t}_i}\right) \tag{24}$$

We note that $\mathbf{w}_i$ is equal to the left singular vector of the empirical covariance $\mathbf{X}_{i-1}\mathbf{Y}^\top$, rather than using (18). Thus, writing as $\mathbf{l}_i$ the corresponding right singular vector, the projections of $\mathbf{X}_i$ and $\mathbf{Y}$ onto $\mathbf{w}_i$ and $\mathbf{l}_i$ respectively have the largest covariance of any possible such projections. This conforms to a common intuition behind PLS in a single dimension, namely that features are chosen so as to maximize covariance between input and output. The deflation procedure in (24) again belies this interpretation, however, in that we do not project the features $\mathbf{w}$ out of $\mathbf{X}$. We again set $\mathbf{W}_{(r)} = \mathbf{W}_r$ after r iterations, and use (17) in predicting $\mathbf{y}^*$ for a test point $\mathbf{x}^*$ (see [20, Theorem 3.2]). The $\mathbf{w}_i$ remain mutually orthogonal (as do the $\mathbf{t}_i$), which can be shown using the same method as in the univariate proof. To our knowledge, however, there is no established link between multivariate PLS and conjugate gradient descent.

2.3.4 Reduced Rank Regression

Where as the previous algorithms select the feature matrix a-priori or in a greedy manner, reduced rank regression (RRR) selects the feature matrix by solving an eigenvalue problem given that the L_2 loss function is used. Suppose $\mathbf{F}^*$ is the ordinary least squares estimate as given in (16) and ${\mathbf{F}_{(r)}}^*$ is the optimal reduced rank solution, then the following equality holds

$$||\boldsymbol{Y} - {\mathbf{F}_{(r)}}^* \boldsymbol{X}||_2^2 = ||\boldsymbol{Y} - \mathbf{F}^* \boldsymbol{X} + (\mathbf{F}^* - {\mathbf{F}_{(r)}}^*)\boldsymbol{X}||_2^2 = ||\boldsymbol{Y} - \mathbf{F}^* \boldsymbol{X}||^2 + ||(\mathbf{F}^* - {\mathbf{F}_{(r)}}^*)\boldsymbol{X}||_2^2,$$

since the space spanned by the least squares error $\boldsymbol{Y} - \mathbf{F}^*\boldsymbol{X}$ is by definition orthogonal to the space which can be spanned by $\boldsymbol{X}$. Therefore, to obtain the best rank constrained

[7] A conceptually similar method to improve performance on collinear data is to use orthogonal features following a pre-whitening step. This is not always numerically stable, however.

predictor in the L_2 sense we have to consider the term $(\mathbf{F}^* - \mathbf{F}_{(r)}{}^*)\boldsymbol{X}$ only. Let the singular value decomposition of $\mathbf{F}^*\boldsymbol{X}$ be $USV^\top$, then we can build the projector

$$P_r = V[\mathbb{1}_{n,r}, 0][\mathbb{1}_{n,r}, 0]^\top V^\top$$

which projects elements in the space $\mathbb{R}^o$ to the subspace corresponding to the first r right singular vectors of the least squares predictions. Using the projector P_r we can construct the optimal rank r predictor in the L_2 sense by $\mathbf{F}_{(r)}{}^* = P_r\mathbf{F}^*$. Unfortunately, this solution relies on the least squares predictor which does not exist for collinear data. Moreover it is only suited for the L_2 loss without any further regularization.

2.4 Multivariate Regression with Stiefel Constraints

We return now to the original multivariate regression setting in section 2.2, and present a direct solution of the optimization problem defined in (1) and (2).

We begin by noting that the alternative statement of the rank constraint (2), which consists in writing the mapping $\mathbf{F}_{(r)}$ in the form (3), still leaves us with a non-trivial optimization problem (7). To see this, let us consider an iterative approach to obtain an approximate solution to 7, by constructing a sequence of predictors $\mathbf{F}_{(r)_1}, \ldots, \mathbf{F}_{(r)_i}$ such that

$$\mathsf{L}(\mathbf{X}, \mathbf{Y}, \mathbf{F}_{(r)_i}) \quad \geq \quad \mathsf{L}(\mathbf{X}, \mathbf{Y}, \mathbf{F}_{(r)_{i+1}}). \tag{25}$$

We might think to obtain this sequence by updating $\mathbf{V}_{i+1}, S_{i+1}$ and $\mathbf{W}_{i+1}$ according to their *free* matrix gradients $\frac{\partial \mathsf{L}}{\partial \mathbf{V}}|_{\theta_i}$, $\frac{\partial \mathsf{L}}{\partial \mathbf{S}}|_{\theta_i}$, and $\frac{\partial \mathsf{L}}{\partial \mathbf{W}}|_{\theta_i}$ respectively, where θ_i denotes the solution $(\mathbf{V}_i, \mathbf{W}_i, S_i)$ at the ith iteration (*i.e.*, the point at which the gradients are evaluated). This is unsatisfactory, however, in that updating $\mathbf{V}$ and $\mathbf{W}$ linearly along their free gradients does *not* result in matrices with orthogonal columns.

Thus, to define a sequence along the lines of (25), we must first show how to optimize over $\mathbf{V}$ and $\mathbf{W}$ in such a way as to retain orthogonal columns. As we saw in section (2.2), the feature matrices are elements on the Stiefel manifold; thus any optimization procedure must take into account the geometrical structure of this manifold. The resulting optimization problem is *non-convex*, since the Stiefel manifold $\mathbb{S}(n, r)$ is not a convex set.

In the next section, we describe how to update $\mathbf{V}$ and $\mathbf{W}$ as we move along geodesics on the Stiefel manifold $\mathbb{S}(n, r)$; in the two sections that follow, we use these updates to conduct the minimization of the L_2 and L_1 losses respectively in the absence of regularization. We introduce regularization for the batch case in section 2.4.4, and describe the on-line algorithm in section 2.4.5.

2.4.1 Dynamics on Stiefel manifolds.

We begin with a description of the geodesics for the simpler case of $\mathbb{S}(n, n)$, followed by a generalization to $\mathbb{S}(n, r)$ when $n > r$. Let $O(n)$ denote the group of orthogonal matrices. Suppose we are given a matrix $\mathbf{V}(t) \in O(n)$ that depends on a parameter t, where $\mathbf{V}(t)$ describes a geodesic on the manifold $O(n)$. Our goal in this subsection is to describe how $\mathbf{V}(t)$ changes as we move along the geodesic. Since $O(n)$ is not only a manifold but also a *Lie group* (a special manifold whose elements form a group), there is an elegant way of moving along geodesics which involves an exponential map.

We will give an informal but intuitive derivation of this map; for a formal treatment, see [16, 23]. We begin by describing a useful property of the derivative of $\mathbf{V}(t)$;

$$\begin{aligned} \mathbb{1} &= \mathbf{V}(t)^\top \mathbf{V}(t), \\ \mathbf{0} &= \frac{d}{dt}(\mathbf{V}(t)^\top \mathbf{V}(t)), \\ \mathbf{0} &= (\frac{d}{dt}\mathbf{V}(t))^\top \mathbf{V}(t) + \mathbf{V}(t)^\top (\frac{d}{dt}\mathbf{V}(t)), \\ \mathbf{0} &= \mathbf{Z}(t)^\top + \mathbf{Z}(t), \end{aligned}$$

with

$$\mathbf{Z}(t) := \mathbf{V}(t)^\top (\frac{d}{dt}\mathbf{V}(t)). \tag{26}$$

The matrix $\mathbf{Z}(t)$ is skew symmetric, which we write as $\mathbf{Z}(t) \in \mathsf{s}(n,n)$, where s consists of the set of all skew symmetric matrices of size $n \times n$.

We next consider curves corresponding to 1-parameter subgroups of $O(n)$; in other words, curves satisfying $\mathbf{V}(0) = \mathbf{I}$ and $\mathbf{V}(t+s) = \mathbf{V}(t)\mathbf{V}(s)$ (in particular, $\mathbf{V}(t)^{-1} = \mathbf{V}(-t)$) for all s, t. For our group $O(n)$, we can obtain such a subgroup by fixing an n-dimensional axis and considering all matrices describing rotations around that axis. In this case, the parameters t, s can be thought of as rotation angles. Returning to (26) with $\mathbf{V}(t)$ in this 1-parameter subgroup, we have

$$\begin{aligned} \mathbf{Z}(t) &= \lim_{\Delta t \to 0} \mathbf{V}(t)^\top \left(\frac{\mathbf{V}(t+\Delta t) - \mathbf{V}(t)}{\Delta t} \right) \\ &= \lim_{\Delta t \to 0} \frac{\mathbf{V}(t)^\top \mathbf{V}(t+\Delta t) - \mathbf{I}}{\Delta t} = \lim_{\Delta t \to 0} \frac{\mathbf{V}(-t)\mathbf{V}(t+\Delta t) - \mathbf{I}}{\Delta t} \\ &= \lim_{\Delta t \to 0} \frac{\mathbf{V}(\Delta t) - \mathbf{V}(0)}{\Delta t} = \frac{d}{dt}\Big|_0 \mathbf{V}(t) = \mathbf{Z}(0), \end{aligned}$$

which means $\mathbf{Z}(t)$ is constant. Multiplying (26) with $\mathbf{V}(t)$ from the left yields an ordinary differential equation of the form

$$\begin{aligned} \frac{d}{dt}\mathbf{V}(t) &= \mathbf{V}(t)\mathbf{Z} \\ \text{with } \mathbf{V}(0) &= \mathbf{V}, \end{aligned} \tag{27}$$

which has solution

$$\mathbf{V}(t) = \mathbf{V}(0)e^{t\mathbf{Z}}, \tag{28}$$

where $e^{\mathbf{Z}}$ denotes the matrix exponential, [34].[8] We can see from (27) that the skew-symmetric matrix $\mathbf{Z}$ specifies a tangent at the point $\mathbf{V}(0)$ on the Stiefel manifold $\mathbb{S}(n,n) = O(n)$.[9]

We now generalize to the case where $\mathbf{V}$ does not have full rank, *i.e.* $\mathbf{V} \in \mathbb{S}(n,r)$ and $r < n$. We can embed $\mathbb{S}(n,r)$ into $O(n)$ by extending any $\mathbf{V} \in \mathbb{S}(n,r)$ with an $n - r$

[8] When verifying that (28) is indeed a solution for (27) note that the skew-symmetric matrix $\mathbf{Z}$ is *normal*, *i.e.* $\mathbf{Z}\mathbf{Z}^\top = \mathbf{Z}^\top\mathbf{Z}$.

[9] Note, moreover, that (27) tells us that every tangent vector of the the curve $\mathbf{V}(t)$ can be obtained by left multiplication of a constant tangent $\mathbf{Z}$ with $\mathbf{V}(t)$. In terms of Lie group theory, the 1-parameter subgroups of a Lie group correspond to "left-invariant" vector fields, and they can be represented using the exponential map (28).

matrix $\mathbf{V}_\perp \in \mathbb{S}(n, n-r)$ such that $\mathbb{R}^n = \mathbf{V}_\perp \bigoplus \mathbf{V}$. Therefore $\mathbf{V}_\perp$ spans the orthogonal complement to the space spanned by the columns of $\mathbf{V}$. Two orthogonal matrices $\mathbf{A}, \mathbf{B}$ in $O(n)$ are considered to be the same from the viewpoint of $\mathbb{S}(n, r)$ if they relate as

$$\mathbf{B} \quad = \quad [\mathbb{1}_{n,r}, \mathbf{P}]\, \mathbf{A} \tag{29}$$

for any matrix $\mathbf{P} \in \mathbb{S}(n, n-r)$. Therefore (27) can be written as

$$\frac{d}{dt}\, [\mathbf{V}(t), \mathbf{V}_\perp(t)]\Big|_{t=0} \quad = \quad [\mathbf{V}(0), \mathbf{V}_\perp(0)]\, \hat{\mathbf{Z}}, \tag{30}$$

and

$$\mathbf{V}(t) \quad = \quad [\mathbf{V}(0), \mathbf{V}_\perp(0)]\, e^{t\hat{\mathbf{Z}}}\, [\mathbb{1}_{n,r}, \mathbf{0}]\,. \tag{31}$$

Note that there is no unique embedding of $\mathbb{S}(n, r)$ to $O(n)$, and our embedding is a specially constructed one.

To conduct a gradient descent procedure, we need to find the tangent direction $\hat{\mathbf{G}}$ (for use in (28)) which is as close as possible to the free gradient $\mathbf{G}$, since $\mathbf{G}$ does not in general have the factorization (27). The constrained gradient $\hat{\mathbf{G}}$ can be calculated directly by projecting the free gradient onto the tangent space $T_{\mathbf{V}}\mathbb{S}(n, r)$ of $\mathbf{V}$; see [23]. Intuitively speaking, we do this by removing the symmetric part of $\mathbf{G}^\top \mathbf{V}$, leaving the skew symmetric remainder. [10] The constrained gradient is thus

$$\hat{\mathbf{G}} \quad = \quad \mathbf{G} - \mathbf{V}\mathbf{G}^\top \mathbf{V}. \tag{32}$$

Finally, the skew symmetric matrix $\hat{\mathbf{Z}} \in \mathbb{R}^{n\times n}$ is given by (26) as

$$\hat{\mathbf{Z}} \quad = \quad \begin{pmatrix} \hat{\mathbf{G}}^\top \mathbf{V} & -(\hat{\mathbf{G}}^\top \mathbf{V}_\perp)^\top \\ \hat{\mathbf{G}}^\top \mathbf{V}_\perp & \mathbf{0} \end{pmatrix}. \tag{33}$$

We can now describe the nonlinear update operator π_{Stiefel}.

2.4.2 Do we need the π_{Stiefel} operation?

A reasonable question to ask is if the gradient descent along the Stiefel manifold is of pure mathematical interest or does one really gain something? The brave could ignore differential geometry and group theory and favor a naive optimization approach where

[10] Any square matrix can be expressed as a unique sum of a symmetric and a skew-symmetric matrix.

Algorithm A2.1 $\pi_{\text{STIEFEL}}(\mathbf{V}, \mathbf{G}, t)$

Given a free gradient $\mathbf{G} \in \mathbb{R}^{n,r}$, an orthogonal matrix $\mathbf{V} \in \mathbb{S}(n, r)$ and a scalar step parameter γ, the update of $\mathbf{V}$ specified by $\mathbf{G}$ and γ can be calculated as follows:

1) Calculate constrained gradient in (32).
2) Calculate orthogonal basis $\mathbf{V}_\perp$ for the orthogonal complement of $\mathbf{V}$.
3) Calculate the tangent coordinates $\mathbf{Z}$ in (33),
4) $\mathbf{V}(\gamma) = [\mathbf{V}, \mathbf{V}_\perp] e^{\gamma \mathbf{Z}} [\mathbb{1}_{n,r}, \mathbf{0}]$.

---

one does a small gradient step and immediately project back to the feasible set of reduced rank predictors. Thus one could calculate the gradient for the L_2 loss

$$\frac{\partial \mathsf{L}_2}{\partial \mathbf{F}}|_{\mathbf{F}_i} = -\mathbf{Y}\mathbf{X}^\top + \mathbf{F}_i\mathbf{X}\mathbf{X}^\top \tag{34}$$

and perform the standard gradient updates

$$\hat{\mathbf{F}}_{i+1} = \mathbf{F}_i - \eta \frac{\partial \mathsf{L}_2}{\partial \mathbf{F}}$$

where we use a fixed learning rate η. Once the update is performed we can project back to the set of reduced rank predictors via the singular value decomposition of $\mathbf{F}$ via

$$[\mathbf{V}, S, \mathbf{W}] = \mathrm{SVD}(\hat{\mathbf{F}}_{i+1}, r), \text{ and thus } \mathbf{F}_{i+1} = \mathbf{V}S\mathbf{W}^\top.$$

Here $\mathrm{SVD}(\mathbf{F}, r)$ yields the r first left and right singular vectors $\mathbf{V}$,$\mathbf{W}$ and a diagonal matrix S with singular values in the diagonal entries.

To demonstrate the different optimization behavior we construct a simple regression problem from $\mathbb{R}^3$ to $\mathbb{R}^3$ with randomly generated input data and a randomly generated predictor $\boldsymbol{F}_(2) \in \mathbb{R}^{3,3}$, subject to rank $\boldsymbol{F}_(2) = 2$. We can try to solve the reduced rank regression problem 1 by this naive procedure. We plot the path of the implicitly defined input and feature matrices $\mathbf{W}, \mathbf{V}$ during optimization, since $\mathbf{W} = [\boldsymbol{w}_1, \boldsymbol{w}_2]$ and $\mathbf{V} = [\boldsymbol{v}_1, \boldsymbol{v}_2]$ consist of 2 vectors in $\mathbb{R}^3$ each. One can see in the top row of figure 2.1 that back projection leads to non-continuous paths of $[\boldsymbol{w}_1, \boldsymbol{w}_2]$ and $[\boldsymbol{v}_1, \boldsymbol{v}_2]$ on the sphere, where as applying π_{Stiefel} to the same problem results in smooth paths of the optimization variable shown in the last row of figure 2.1. Although a standard gradient descent leads to a decrease of the residual error, it is likely that back-projection compensates most of this improvement since they work independently from each other. Indeed, back-projection is completely independent on the state before the gradient step and on the objective function of the problem. In figure 2.1 one can see where the path penetrates the sphere that gradient descent and back-projection can even lead to oscillation on the feasible set. On the other hand, the π_{Stiefel} operation takes the original gradient and solves a motion equation that ensures that we stay at all times on the feasible set. However, one could argue that one might not be interested in the path of the feature matrices, since what finally counts is the minimization of the objective function (1). For this reason let us compare the minimization performance of both approaches. One can see in figure 2.2 that gradient descent on Stiefel manifold clearly outperforms the direct gradient descent approach. Since we clearly demonstrated the necessity for the differential geometrical update operator π_{Stiefel} we are ready to motivate our regression algorithm.

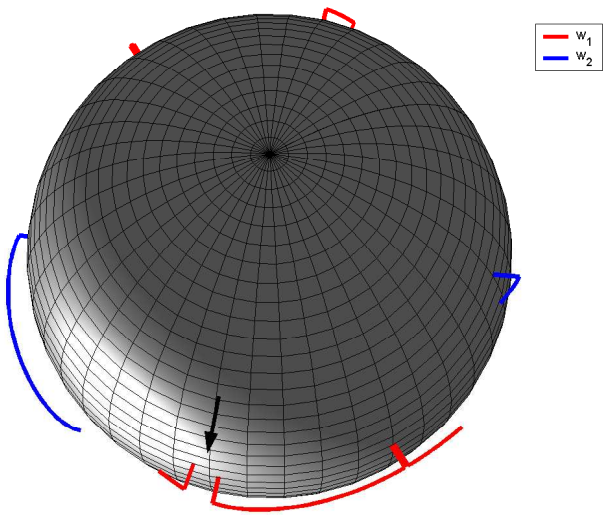
2.4.3 Multivariate regression with L_2 loss

We can apply a gradient descent approach to (7) which uses the π_{Stiefel} operation to perform the descent. We first calculate the free gradients,

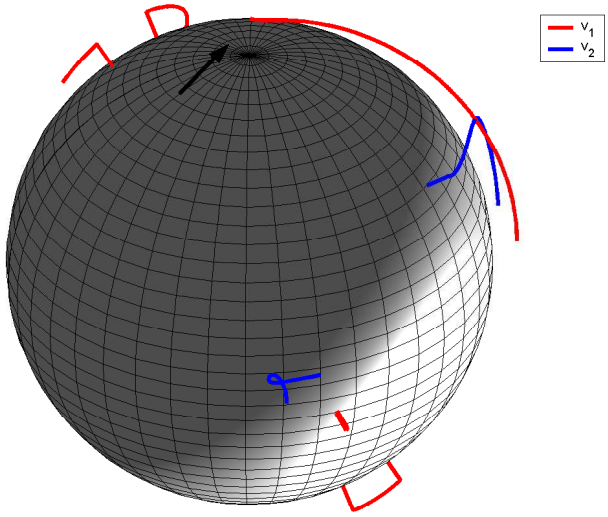
$$\frac{\partial \mathsf{L}_2}{\partial \mathbf{V}}|_{\theta_i} = -\mathbf{Y}\mathbf{X}^\top \mathbf{W}_i S_i, \tag{35}$$

$$\frac{\partial \mathsf{L}_2}{\partial \mathbf{W}}|_{\theta_i} = -\mathbf{X}\mathbf{Y}^\top \mathbf{V}_i S_i + \mathbf{X}\mathbf{X}^\top \mathbf{W}_i S_i^2, \tag{36}$$

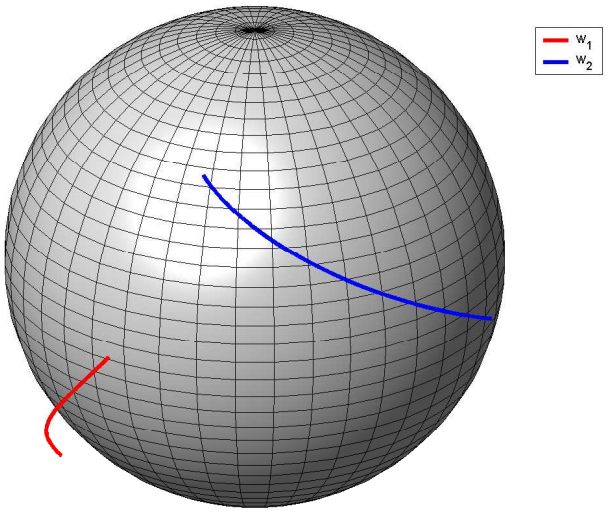
$$\frac{\partial \mathsf{L}_2}{\partial S}|_{\theta_i} = -\mathbb{1}_{r,r} \odot \mathbf{W}_i^\top \mathbf{X}\mathbf{Y}^\top \mathbf{V}_i + \mathbb{1}_{r,r} \odot \mathbf{W}_i^\top \mathbf{X}\mathbf{X}^\top \mathbf{W}_i S_i, \tag{37}$$



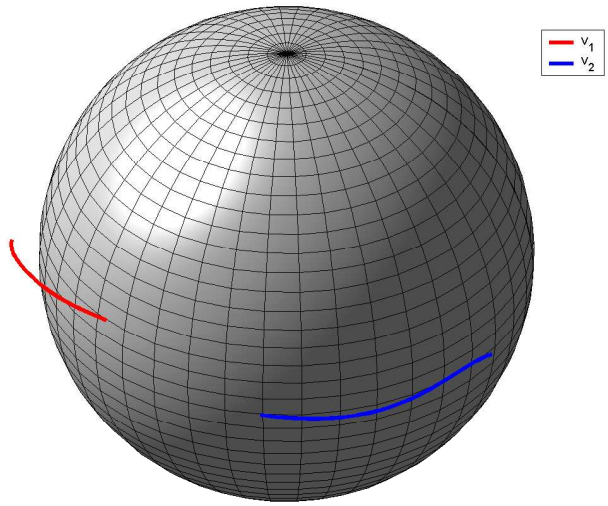
Path of $\mathbf{W}$ using standard gradient descent.



Path of $\mathbf{V}$ using standard gradient descent.



Path of $\mathbf{W}$ using π_{Stiefel}.



Path of $\mathbf{V}$ using π_{Stiefel}.

Figure 2.1: Paths of optimization variables using standard additive gradient descent and exponential gradient update using π_{Stiefel}. Top row: SVD back projection performs variables to *jump* on the sphere surface. Jump positions are indicated by arrows. Bottom row: Paths done by exponential gradient are smooth.

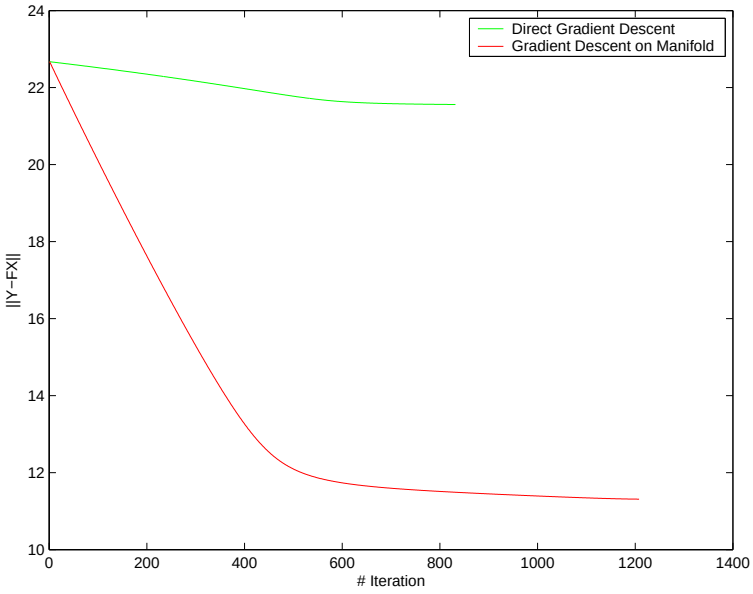


Figure 2.2: Minimization performance of direct gradient descent vs gradient descent on the Stiefel manifold using π_{Stiefel}.

where $\odot$ denotes the Hadamard (element-wise) product. The free gradients are then projected on the tangent space and used afterwards for a line search along geodesics using the π_{Stiefel} operator. The multivariate regression algorithm for the L_2 loss is summarized in detail in A2.2. Note that alternative to the line search procedure, one can also use a fixed learning rate η which would replace step 2 in algorithm A2.2.

2.4.4 Soft-Regularization

So far we only control the number of parameters via the rank constrained but not the magnitude of the parameters. Since we have formulated our learning problem as an optimization task it is quite easy to incorporate an additional regularizer Ω_d which penalizes the squared norm $||\mathbf{F}_{(r)}||^2$ (in a manner analogous to ridge regression), since $||\mathbf{F}_{(r)}||_F^2 = ||S||_F^2$. The regularized version of (7) is

$$\theta \;=\; \underset{\mathbf{F}_{(r)}=\mathbf{V}S\mathbf{W}^\top}{\arg\min} \|\mathbf{Y}-\mathbf{V}S\mathbf{W}^\top\mathbf{X}\|_F^2 + \lambda\|S\|_F^2. \tag{38}$$

subject to constraints (4)-(6). The only change in algorithm A2.2 is to replace the gradient $\frac{\partial \mathsf{L}_2}{\partial S}|_\theta$ by

$$\frac{\partial \hat{\mathsf{L}}_2}{\partial S}|_\theta \;=\; \frac{\partial \mathsf{L}_2}{\partial S}_\theta + \lambda S. \tag{39}$$

2.4.5 On-line variant: S-MRS

In this section, we describe the sequential implementation of MRS, denoted by S-MRS. The main difference, compared with the batch method, is the addition to the loss of a regularizing functional [11] Ω_o, which controls the trade-off between fit to the newly

[11] This should not be confused with the regularizer Ω_d introduced in the previous section, which controls $||\mathbf{F}_{(r)}||^2$ for a solution obtained with a particular batch of data.

Algorithm A2.2 MRS FOR L_2 LOSS FUNCTION

Initialization

$\mathbf{V}_0 = \mathbb{1}_{d,r} \quad S_0 = \mathbb{1}_{r,r} \quad \mathbf{W}_0 = \mathbb{1}_{o,r}$

$\theta_0 = (\mathbf{V}_0, S_0, \mathbf{W}_0) \quad \mathbf{F}_{(r)_0} = \mathbf{W}_0 S_0 \mathbf{V}_0 \quad i = 0$

Repeat until convergence:

1) Calculate free gradients, equations (35)-(37)
2) $t^*_{\mathbf{V}}, t^*_S, t^*_{\mathbf{W}} = \underset{t_{\mathbf{V}}, t_S, t_W}{\arg\min} \mathsf{L}_2(\mathbf{V}(t_{\mathbf{V}}), S(t), \mathbf{W}(t_{\mathbf{W}}))$
   with $\mathbf{W}(t_{\mathbf{W}}) = \pi_{stiefel}(\mathbf{W}_i, \frac{\partial \mathcal{L}}{\partial \mathbf{W}}|_{\theta_i}, t_{\mathbf{W}})$,
   $\mathbf{V}(t_{\mathbf{V}}) = \pi_{stiefel}(\mathbf{V}_i, \frac{\partial \mathcal{L}}{\partial \mathbf{V}}|_{\theta_i}, t_{\mathbf{V}})$,
   and $S(t) = S_i + t_S \frac{\partial \mathcal{L}}{\partial S}|_{\theta_i}$
3) $\mathbf{V}_{i+1} = \mathbf{V}(t^*_{\mathbf{V}}), \quad \mathbf{W}_{i+1} = \mathbf{W}(t^*_{\mathbf{W}}), \quad S_{i+1} = S(t^*_S)$
4) $\mathbf{F}_{(r)_{i+1}} = \mathbf{V}_{i+1} S_{i+1} \mathbf{W}_{i+1}$
5) $\theta_{i+1} = (\mathbf{V}_{i+1}, S_{i+1}, \mathbf{W}_{i+1})$
6) $i = i + 1$

After convergence : $\mathbf{F}_{(r)} = \mathbf{F}_{(r)_i}$

observed data block $\mathbf{Z}_k = [\mathbf{X}_k, \mathbf{Y}_k]$ (as measured by L_2), and distance to the predictor $\mathbf{F}^{(k-1)}_{(r)}$ obtained from previous data $\mathbf{Z}_j$ with $j < k$ (see (9) in section 2.2.1 for more detail). Thus the regularization functional takes the form

$$\Omega_o\left(\mathbf{V}_{(r)} S_{(r)} {\mathbf{W}_{(r)}}^\top, \mathbf{V}^{(k-1)}_{(r)} S^{(k-1)}_{(r)} {\mathbf{W}^{(k-1)}_{(r)}}^\top\right) \;=\; \gamma \left\| \mathbf{V}_{(r)} S_{(r)} {\mathbf{W}_{(r)}}^\top - \mathbf{V}^{(k-1)}_{(r)} S^{(k-1)}_{(r)} {\mathbf{W}^{(k-1)}_{(r)}}^\top \right\|^2 .$$

Since Ω_o (and thus $\mathcal{L}$) is differentiable, the free gradients $\frac{\partial \mathcal{L}}{\partial \mathbf{V}}|_{\theta_i}, \frac{\partial \mathcal{L}}{\partial \mathbf{W}}|_{\theta_i}$ and $\frac{\partial \mathcal{L}}{\partial S}|_{\theta_i}$ can be calculated analytically.

The regularization parameter γ controls convergence behavior. For example, we can increase γ for every new block k to progressively decrease the influence of new observations. On the other hand, a fixed γ can be used to implement adaptive behavior if the distribution generating the observations changes over time. Given these quantities, and starting with $\mathbf{F}_{(r)} = \mathbb{1}$ at $k = 0$, we can adapt algorithm A2.2 to solve (10) for each $k > 0$ and associated observation sequence $\mathbf{X}_k, \mathbf{Y}_k$. The on-line algorithm given is described in A2.3.

All techniques introduced so far use a gradient descent type procedure on the quantities $\mathbf{W}, \mathbf{V}$ and S. Thus, in principle we are using a *path-following* optimization algorithm starting at a particular point and are likely to end at some local minima. To this reason, in the next section we reconsider the inference problem in the Bayesian framework which provides us with an optimal distribution over the space of predictors. We will exploit this theoretical optimal distribution to generate predictors via sampling using Markov Chain Monte Carlo (MCMC) sampling schemes. Later in chapter 3 we will reconsider a similar approach to hard optimization problems in a more general perspective.

Algorithm A2.3 SEQUENTIAL MULTIVARIATE REGRESSION WITH STIEFEL CONSTRAINTS

Given data $\mathbf{X}_k, \mathbf{Y}_k$, a learning rate γ and a reference mapping $\mathbf{F}_{(r)}^{(k-1)} = \mathbf{V}_{(r)}^{(k-1)} S_{(r)}^{(k-1)} \mathbf{W}_{(r)}^{(k-1)\top}$

Initialization

$\mathbf{V}_0 = \mathbf{V}_{(r)}^{(k-1)} \quad S_0 = S_{(r)}^{(k-1)} \quad \mathbf{W}_0 = \mathbf{W}_{(r)}^{(k-1)}$

$\theta_0 = (\mathbf{V}_0, S_0, \mathbf{W}_0) \quad \mathbf{F}_0 = \mathbf{W}_0 S_0 \mathbf{V}_0 \quad i = 0$

Repeat until convergence:

1) Calculate free gradients $\frac{\partial \mathcal{L}}{\partial \mathbf{V}}|_{\theta_i}$, $\frac{\partial \mathcal{L}}{\partial \mathbf{W}}|_{\theta_i}$ and $\frac{\partial \mathcal{L}}{\partial S}|_{\theta_i}$.
2) $t_{\mathbf{V}}^*, t_S^*, t_{\mathbf{W}}^* = \underset{t_{\mathbf{V}}, t_S, t_W}{\arg\min} \mathcal{L}(\mathbf{Z}_k, \mathbf{V}(t_{\mathbf{V}})\, S(t_S)\, \mathbf{W}(t_{\mathbf{W}}), \mathbf{F}_{(r)}^{(k-1)})$
   with $\mathbf{W}(t_{\mathbf{W}}) = \pi_{stiefel}(\mathbf{W}_i, \frac{\partial \mathcal{L}}{\partial \mathbf{W}}|_{\theta_i}, t_{\mathbf{W}})$,
   $\mathbf{V}(t_{\mathbf{V}}) = \pi_{stiefel}(\mathbf{V}_i, \frac{\partial \mathcal{L}}{\partial \mathbf{V}}|_{\theta_i}, t_{\mathbf{V}})$,
   and $S(t) = S_i + t_S \frac{\partial \mathcal{L}}{\partial S}|_{\theta_i}$
3) $\mathbf{V}_{i+1} = \mathbf{V}(t_{\mathbf{V}}^*), \quad \mathbf{W}_{i+1} = \mathbf{W}(t_{\mathbf{W}}^*), \quad S_{i+1} = S(t_S^*)$
4) $\mathbf{F}_{i+1} = \mathbf{V}_{i+1} S_{i+1} \mathbf{W}_{i+1}$
5) $\theta_{i+1} = (\mathbf{V}_{i+1}, S_{i+1}, \mathbf{W}_{i+1})$
6) $i = i + 1$

After convergence : $\mathbf{F}_{(r)} = \mathbf{F}_i$, $k = k + 1$

2.5 The probabilistic viewpoint – Bayesian Inference with rank constraint

Instead of solving the non-convex problem (1) via a gradient-descent-based procedure, we can try to avoid it entirely via probabilistic methods: If we were able to define a distribution $P(\mathbf{F}_{(r)})$ over all possible predictors such that *good* predictors have high probability, we could generate samples from this distribution. As a consequence, sampling from this distribution is likely to generate good predictors and requires us to consider two issues:

1. What are useful distributions with high probability measure on *good* predictors?
2. How can we sample predictors $\mathbf{F}_{(r)}$ from such a distribution?

Let us start our discussion by addressing the first point.

2.5.1 The Anatomy of the Posterior Distribution

We assume that predictors $\mathbf{F}_{(r)}$, input data $\boldsymbol{X}$ and output data $\boldsymbol{Y}$ have a joint distribution that depends on a set of additional hyperparameters τ. We are interested on the *posterior* probability of a predictor $\mathbf{F}_{(r)}$, given $\boldsymbol{X}, \boldsymbol{Y}$ and τ. Using the Bayes formalism, we can create a posterior distribution over the set of predictors, given a cost function or a likelihood function and any prior knowledge about the solution. Let us denote this posterior by $P(\mathbf{F}_{(r)}|\boldsymbol{X}, \boldsymbol{Y}, \tau)$ describing the dependence of $\mathbf{F}_{(r)}$ on the data $\boldsymbol{X}, \boldsymbol{Y}$ and hyperparameters τ. If we assume that $\mathbf{F}_{(r)}$, τ and $\boldsymbol{X}$ are independent and $\boldsymbol{X}$ has a flat prior, this posterior

factorizes as follows

$$P(\mathbf{F}_{(r)}|\boldsymbol{X},\boldsymbol{Y},\tau) \propto P(\boldsymbol{Y}|\mathbf{F}_{(r)},\boldsymbol{X},\tau)P(\mathbf{F}_{(r)})P(\tau), \tag{40}$$

where $P(\boldsymbol{Y}|\mathbf{F}_{(r)},\boldsymbol{X},\tau)$ denotes the user-provided likelihood function with hyperparameter τ and where $P(\mathbf{F}_{(r)})$ denotes a prior probability over the set

$$\mathcal{H}_{(r)} := \left\{\mathbf{F}_{(r)} \in \mathbb{R}^d \to \mathbb{R}^o \,|\, \mathbf{rank}\ \mathbf{F}_{(r)} = r\right\}. \tag{41}$$

Thus, once we have specified our likelihood model and our prior assumptions in terms of parametric distribution functions, we are ready to generate samples from the posterior. The appealing feature of the probabilistic approach is versatility. We can use any likelihood function, i.e. any cost function we want. A common example is the Gaussian likelihood which corresponds to the L_2 error:

$$P(Y|\mathbf{F}_{(r)},X,\tau) \propto e^{-\frac{1}{2\tau^2}||Y-\mathbf{F}_{(r)}X||^2}.$$

We can use even non-differentiable cost functions by constructing the corresponding likelihood function. Consider the case that our data is corrupted by noise from a heavy tailed distribution. In such a situation the Laplacian likelihood,

$$P(Y|\mathbf{F}_{(r)},X,\tau) \propto e^{-\frac{1}{\tau}||Y-\mathbf{F}_{(r)}X||},$$

can be used. This corresponds to using the non-differentiable L_1 loss function. After having defined our likelihood function, we have to define priors appropriate for our model. Let us start with the prior $P(\mathbf{F}_{(r)})$. Since $\mathbf{F}_{(r)}$ factorizes as

$$\mathbf{F}_{(r)} = \mathbf{V}_{(\mathbf{r})}S\mathbf{W}_{(\mathbf{r})}^{\top},$$

we have to define a prior $P(\mathbf{V}_{(\mathbf{r})},\mathbf{S},\mathbf{W}_{(\mathbf{r})})$. If we assume that $\mathbf{V}_{(\mathbf{r})}S\mathbf{W}_{(\mathbf{r})}^{\top}$ are independent, the prior is of the form

$$P(\mathbf{F}_{(r)}) = P(\mathbf{V}_{(\mathbf{r})})P(S)P(\mathbf{W}_{(\mathbf{r})}).$$

Using this prior we obtain the final form of our a posteriori function:

$$P(\mathbf{F}_{(r)}|\boldsymbol{X},\boldsymbol{Y},\tau) \propto P(\boldsymbol{Y}|\mathbf{F}_{(r)},\boldsymbol{X},\tau)P(\mathbf{V}_{(\mathbf{r})})P(S)P(\mathbf{W}_{(\mathbf{r})})P(\tau). \tag{42}$$

We give $\mathbf{V}_{(r)}$ and $\mathbf{W}_{(r)}$ uniform distributions over the Stiefel manifold, since in general no $\mathbf{V}_{(r)},\mathbf{W}_{(r)}$ can be preferred over another. For the entries of the diagonal matrix S we would like to use a prior which is invariant under linear transformation, i.e. $P(S) = P(cS)$, since we do not know in advance the scale of the entries of S. However, we know a priori that only positive values are allowed. Both of these requirements are met by the so-called *Jeffries prior* (see e.g. [57], chapter 23), $p(S_i) = 1/S_i\mathbb{1}_{S_i>0}$ ($\mathbb{1}$ is the indicator function). We now have all the components required to specify the posterior (up to normalization). For both possible likelihood functions, the model parameter τ can be interpreted as some measure of the amount of noise. Since we do not have any knowledge about the magnitude of the noise as well, we use again the scale invariant Jeffries prior for τ.

Let us now come to the second point: How can we sample from the distribution in (40)?

2.5.2 Sampling from the Posterior using MCMC

Let us briefly discuss how sampling using Markov Chain Monte Carlo (MCMC) works in general and afterwards proceed with the description of the details for our specific case. Consider an arbitrary distribution function $P(Z)$ such that it is not possible to directly generate samples from it. The most naive way to generate samples from $P(Z)$ is the acceptance-rejection rule:[12]

1. Draw a random sample Z according to a uniform distribution.
2. If $P(Z)$ is larger than a random threshold drawn uniformly from the unit interval, accept this as sample from $P(Z)$, otherwise reject.

Although straight forward in its implementation, the acceptance-rejection rule is suboptimal. To see this consider $P(Z)$ to have a compact support. Note that the acceptance-rejection rule generates samples $Z_1, Z_2, Z_3, \ldots$ independent from previous success or failure, i.e. this sampling process does not have a memory. Therefore, since all points outside the support of $P(Z)$ have probability zero, the acceptance-rejection rule is likely to generate repeatedly samples Z that will be rejected. This effect becomes more prnonounced for high-dimensional problems, since the probability that a sample will fall outside of the support of $P(Z)$ will increase. The idea of MCMC methods is to replace this independent sampling process by one which has a memory. To this end, one uses a Markov process which is determined by the transition equation

$$Z^{(t+1)} \sim T(Z^{(t+1)}|Z^{(t)}),$$

and an initial distribution function $T_0(Z)$.[13] Thus, if a state is in a high probability region, it is likely that the next state will have high probability as well. The question is, how to design the transition function $T(Z^{(t+1)}|Z^{(t)})$ such that the equilibrium distribution – ideally our distribution $P(Z)$ – is reached as quickly as possible. Furthermore, to be of any benefit, the construction of the next state $Z^{(t+1)}$ of the Markov process given the current state $Z^{(t)}$ must be efficient. To this end, besides leading to quick convergence, the transition rule must be easy to evaluate as well. The Metropolis-Hastings (MH) algorithm prescribes exactly such a transition rule. It requires to specify an *arbitrary* proposal distribution function $q(\hat{Z}^{(t+1)}|Z^{(t)})$ which generates candidates $\hat{Z}^{(t+1)}$ given the current state $Z^{(t)}$. Ideally, the proposal distribution q would generate candidates that are in high-density regions of $P(Z)$. The choice of the proposal distribution function is critical for the resulting convergence speed and we will discuss later proposal functions for our problem in more detail. Using candidates generated by the proposal function, the MH rule leads to a Markov process which reaches any target distribution independent $P(Z)$ of a starting distribution $T_0(Z_0)$. We give a description of the MH algorithm in A2.4. For a rigorous review and analysis see [67].

Now, note that so far we have discussed sampling from one random variable Z. In the case of rank-reduced regression, we have multiple random variables $\theta = \{\mathbf{V}, S, \mathbf{W}, \tau\}$ and not just a single one. In principle, we could stack all variables to one random vector Z. This would require us to define a single proposal function $q(\hat{Z}|Z)$ to generate new candidates for Z. However, in our setting, it is easier to define proposal functions for all

[12] For simplicity we discuss the case that one draws Z from a uniform distribution. This is the so-called proposal distribution, and ideally one chooses a proposal distribution being as close to P as possible.

[13] Since in general, one does not have a prior on the distribution of Z in space one assumes the initial distribution $T_0(Z)$ to be uniform.

Algorithm A2.4 METROPOLIS-HASTINGS METHOD FOR DRAWING SAMPLES FROM $P(Z)$

1) Sample $u \sim \mathcal{U}[0,1]$, where $\mathcal{U}[0,1]$ uniform on $[0,1]$
2) Sample $\hat{Z}_k \sim q(\hat{Z}^{(k)}|Z^{(k)})$, where $q(\hat{Z}^{(k)}|Z^{(k)})$ is the proposal dependent on the current state $Z^{(k)}$.
3) Set $\alpha(Z^{(k)}, \hat{Z}^{(k)}) := \min\left[1, \frac{Q(\hat{Z}^{(k)})q(Z^{(k)}|\hat{Z}^{(k)})}{Q(Z^{(k)})q(\hat{Z}^{(k)}|Z^{(k)})}\right]$
4) If $u < \alpha(Z^{(k)}, \hat{Z}^{(k)})$ $\quad Z^{(k+1)} := \hat{Z}^{(k)}$, otherwise $Z^{(k+1)} := Z^{(k)}$

variables independently. [14] A sampling strategy which allows for the construction of a Markov chain for each model variable independently is *Gibbs* sampling. The advantage of a Gibbs sampler is that it alters only a single variable per state transition. Thus, it can be understood as a coordinate-wise sampling strategy. However, the only requirement of Gibbs sampling is that each variable is drawn from its *conditional* distribution. Let us briefly describe Gibbs sampling.

1. Select a single element $\theta_i^{(k)} \in \theta^{(k)}$
2. Update this single element with a new sample drawn from the conditional distribution $\pi\left(\theta_i^{(k)} \middle| \theta^{(k)} \setminus \{\theta_i^{(k)}\}\right)$ and leave the remaining components unchanged.
3. Set $k = k+1$ and $i = i+1$. If $i > |\theta^{(k)}|$, then $i = 1$.

Unfortunately, in our case it is also difficult to sample from the conditional distribution of each variable. However, we can use the Metropolis-Hastings algorithm for this task. Thus, if we use the MH algorithm inside the Gibbs sampling for the conditional distribution $\pi\left(\theta_i^{(k)} \middle| \theta^{(k)} \setminus \{\theta_i^{(k)}\}\right)$, we obtain the so-called *Metropolized Gibbs* sampler. In this work, we will apply this sampler since our model consists of four independent variables. Before we finally apply the Metropolized Gibbs sampler, let us give an example how the variables $\theta = \{\mathbf{V}, S, \mathbf{W}, \tau\}$ of our model would evolve while sampling.

Example E2.1 Variables while performing a cycle of Metropolis Gibbs sampling.: *Let $\theta^0 = \{\mathbf{V}^0, S^0, \mathbf{W}^0, \tau^0\}$ be the start state. We assume that we are given three proposal distributions $q_{\mathbf{V}}, q_S, q_{\mathbf{W}}$ for the MH algorithm.*

Repeat

$\mathbf{V}^i \overset{MH}{\rightarrow} \mathbf{V}^{i+1}$, *given* $\theta = \{\mathbf{V}^i, S^i, \mathbf{W}^i, \tau^i\}$,

$\mathbf{W}^i \overset{MH}{\rightarrow} \mathbf{W}^{i+1}$, *given* $\theta = \{\mathbf{V}^{i+1}, S^i, \mathbf{W}^i, \tau^i\}$,

$S^i \overset{MH}{\rightarrow} S^{i+1}$, *given* $\theta = \{\mathbf{V}^{i+1}, S^i, \mathbf{W}^{i+1}, \tau^i\}$,

$\tau^i \overset{MH}{\rightarrow} \tau$, *given* $\theta = \{\mathbf{V}^{i+1}, S^{i+1}, \mathbf{W}^{i+1}, \tau^i\}$

$i = i+1$

[14] In fact, for MCMC methods, it is of advantage to exploit the decomposition into quantities living in lower-dimensional spaces since the lower dimensional the state variables are the faster convergence will happen. [67].

Once we define proposal distributions for each variable of our model we can utilize Metropolized Gibbs sampling to generate predictors according to the posterior. The most difficult aspect is how to choose a proposal distribution determines the sampling process for $\mathbf{V}_{(r)}$ and $\mathbf{W}_{(r)}$ for the Metropolis steps. Let us investigate some possible approaches. For the case of the orthogonal group, a straightforward method would be to use elementary rotation matrices parameterized by Euler angles around some rotation axes e_i. Since any element of the orthogonal group can be written as a matrix product of elementary rotations, sampling over Euler angles is indeed a valid approach. There is a problem with this direct approach, however: for rank deficient matrices on the Stiefel manifold, we would need to sample both over the rotation angles *and* over rotation axis. To see this, consider sampling from the Stiefel manifold $\mathbb{S}(3,2)$. In this case, we would have to first set two rotation axes in a three-dimensional space and then draw two Euler angles to determine an element from $\mathbb{S}(3,2)$. Obviously, sampling the Euler angles alone is not sufficient since one has to specify the corresponding axis first. For this reason, Euler angles can only be used in obtaining elements from the orthogonal group $O(n) = \mathbb{S}(n,n)$ since all rotation axis are defined in advance. An additional difficulty arises since the matrix product is non-commutative, which implies that using a specific rotation angle parametrization does not encode the elements of the Stiefel manifold in a unique way. This would increase the variance of the feature matrix samples $\mathbf{W}, \mathbf{V}$ when they are encoded in this manner.

Another way to generate proposals on the Stiefel manifold would be to directly apply an orthogonalization procedure to a set of r random points in $\mathbb{R}^n$; see [29, 89]. This procedure results in proposals taken from a uniform distribution over the Stiefel manifold. However, it is well known (e.g. [67]) that the Markov chain resulting from an MH step with this proposal, which takes no account of the previous sample, can take a long time to reach its equilibrium distribution. A more useful proposal for MH takes as a starting point the value of the current state and performs some perturbation to generate the candidate for the next state. A conceptually pleasing distribution for this perturbation could take an equivalent form to the normal distribution centered at the present sample point, but on the Stiefel manifold. The counterpart of a normal distribution on the Stiefel manifold is the Langevin (or von Mises-Fisher) distribution, which for a random matrix $X \in \mathbb{S}(n,r)$ takes the form

$$P(X|F,\sigma^2) \doteq \frac{1}{Z} e^{\frac{1}{\sigma^2}\operatorname{tr}(F^\top X)},$$

where $Z = \int_X f(X)\mu(X)$ according to the probability measure $\mu : \mathbb{S}(n,r) \to \mathbb{R}$, and F the point on $\mathbb{R}^r_+$ equivalent to the mean; see [15].[15] Sampling from the Langevin distribution cannot be carried out directly and thus requires an additional sampling scheme — this is inefficient and should be avoided.

The proposal distribution we use in practice is in fact similar to the Langevin distribution, but can be constructed using the geodesic flow operator introduced in algorithm A2.1. The proposal first requires sampling from a normal distribution $\mathcal{N}(\mathbf{0}; \sigma^2 \mathbb{1})$ on the tangent space $T_F\mathbb{S}(n,r)$ of a point $F \in \mathbb{S}(n,r)$. For the case of the orthogonal group, our true proposal distribution has the form

$$P_l(X|F,\sigma^2) \propto e^{-\frac{1}{2\sigma^2}||\operatorname{logm}(F^\top X)||_F^2}, \tag{43}$$

where we have inverted the exponential mapping to map a point X from the Stiefel manifold back to the tangent space of a particular point F. To see this, consider two points F, X on the orthogonal group. We know that one can be obtained by the other by

[15] For the Stiefel manifold, one can compute Z analytically.

Algorithm A2.5 SAMPLING USING UNIFORM DISTRIBUTION ON THE STIEFEL MANIFOLD $\mathbb{S}(n, r)$

1) Construct a matrix $N \in \mathbb{R}^{n \times r}$, with each element sampled from a zero mean unit variance Gaussian $\mathcal{N}(0, 1)$.
2) Apply a *thin* QR Decomposition of N to construct Q and R.
3) Let D be a diagonal matrix with $D_{ii} = \mathbf{sign}(R_{ii})$
4) Construct $\hat{Q} = QD$.

the geodesic flow operator as

$$X = Fe^Z,$$

where Z encodes the *direction*. If we multiply by $F^\top$ and invert the matrix exponential operation, we obtain the skew symmetric matrix Z, which is always possilbe in the vicinity of F. This can be interpreted as coordinates of X in the (linear) tangent space of F. If the Frobenius norm of Z is large, X will be further apart from F. This distance is the argument of the Gaussian distribution in (43).

Using the equation for the geodesic trajectory (28) we can easily generate variates $X \in \mathbb{S}(n, r)$ from this distribution via

$$X = [F, \overline{F}]e^Z[\mathbb{1}_{n,r}, 0]^\top,$$

where $Z \in \mathfrak{s}(n)$ is a random skew symmetric matrix of size $n \times n$ with components drawn from $\mathcal{N}(\mathbf{0}; \sigma^2)$. For the case of $\mathbb{S}(2, 1)$ (the unit circle in a plane) our proposal distribution is illustrated in figure 2.3, and details of the proposed sampling scheme are described in algorithm A2.6.

We now briefly describe the proposal on the scaling matrix S, with reference to a particular entry S_i. Since we require that any S_i remains positive, we make the proposal

$$S_i^* = S_i e^\lambda, \tag{44}$$

where $\lambda \sim \mathcal{N}(0, \eta\mathbb{1})$ and $\eta \in \mathbb{R}$ takes on the role of an exploration rate. This corresponds to a linear proposal on the logarithm of the S_i, i.e.

$$\log(S_i^*) = \log(S_i) + \lambda.$$

It follows that the proposal for S_i is

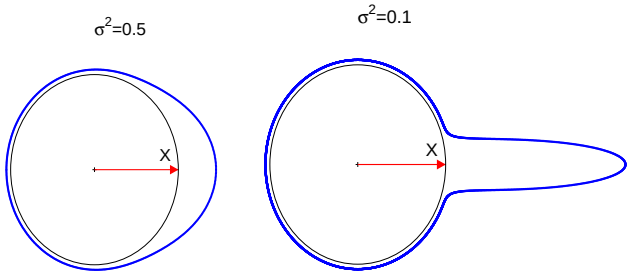
$$q(S_i^*|S_i) \propto \exp\left(-\frac{1}{2\eta^2}\left\|log(S_i^*) - log(S_i)\right\|^2\right)$$

We use an identical method to propose new samples for τ.

We are now all set and can describe the probabilistic version of multivariate regression using sampling. The resulting algorithm is given in A2.7. To determine if our sampling-based algorithm has converged, we propose to perform N_1 steps to overcome the so-called *burn-in* phase. Thus the user has to specify the parameter N_1 in advance, and the algorithm will return N_2 predictors *after* first sampling N_1 times. The parameter N_1 can be understood as the time the Markov process needs to reach equilibrium. For more advanced convergence controls see [57, 67].

Algorithm A2.6 SAMPLING USING A GAUSSIAN ON THE TANGENT SPACE OF AN ELEMENT X OF THE STIEFEL MANIFOLD $\mathbb{S}(n,r)$

1) Construct a matrix $\hat{\theta} \in \mathbb{R}^{n\times n}$, with each element sampled from a zero mean Gaussian $\mathcal{N}(0,\sigma^2)$.
2) Take $\theta = \frac{1}{2}(\hat{\theta} - \hat{\theta}^\top)$
3) Construct $\hat{Q} = [X, \overline{X}]e^{\theta}[\mathbb{1}_{n,r}, 0]^\top$



a) b)

Figure 2.3: Illustration of proposal distribution for the case of $\mathbb{S}(2,1)$. The circle in a) and b) has radius 0.95. The probability $P(Z|X,\sigma^2)$ is calculated by evaluating a standard multivariate Gaussian distribution on the tangent space at X.

2.6 Experiments

In this section we evaluate the performance of the introduced algorithms in various scenarios. We start with a comparison of the introduced deterministic algorithm 2.4 against Partial Least Squares which currently is the standard method for reduced rank regression.

2.6.1 Comparing PLS and MRS-L_2

We compare the introduced deterministic algorithm MRS with PLS in a systematic way and test if their performance differs significantly. Our aim is to demonstrate that, since PLS regularizes via early stopping, MRS will mostly outperform PLS. To create a systematic test, we generate data points $X_i \in \mathbb{R}^{20}$ by sampling from a zero mean Gaussian distribution with covariance matrix $C_{xx}(\kappa) \in \mathbb{R}^{20\times 20}$ which dependents on the scalar κ. To this end, we first generate a random orthogonal basis in $\mathbb{R}^{20}$ for $n = 20, r = 20$ and scale the i-th vector by $s_i = e^{\left(-\frac{i}{2^\kappa}\right)}$. The desired random sample $X \in \mathbb{R}^{20\times N}$ can now be generated by sampling from a multivariate Gaussian and projecting onto this basis. In this way, we generate a rich variety on possible linear regression problem with different data collinearity conditions under the assumption that the input data is Gaussian distributed and the noise is Gaussian.

For each rank constraint, and each κ we generate a regression problem with $y_i = \mathbf{F}^0 X_i + \epsilon$, where $\mathbf{F} \in \mathbb{R}^{20\times 20}$ is a random linear transformation and perturb the linear transformation $\hat{y}_i = \mathbf{F}X_i$ by a random vector $\epsilon \in \mathbb{R}^{20}$ which is sampled from a Gaussian distribution $\mathcal{N}(0;\sigma^2)$. For κ we choose $1,\ldots,14$ since for $\kappa \geq 15$ the spectra are numerically identical. Thus for each of the 14 regression scenarios we generate 20 different rank

Algorithm A2.7 PROBABILISTIC MRS

Probabilistic MRS to obtain the MAP Estimate given a posterior distribution

$$P(\mathbf{F}_{(r)}|Y,X,\tau) \propto P(Y|\mathbf{F}_{(r)},X,\tau)\pi(\tau)$$

and uniform prior over $\mathcal{M}_\theta$.

Initialization

$$\mathbf{V}_0 = \mathbb{1}_{d,r} \quad S_0 = \mathbb{1}_{r,r} \quad \mathbf{W}_0 = \mathbb{1}_{o,r}$$

$$\theta_0 = (\mathbf{V}_0, S_0, \mathbf{W}_0) \quad \mathbf{F}_{(r)_0} = \mathbf{V}_0 S_0 \mathbf{W}_0^\top$$

For $k = 1 \dots N_1 + N_2$

1) Apply algorithm A2.4 to $\hat{\mathbf{W}}$ and with proposal given by algorithm A2.6.
2) Apply algorithm A2.4 to $\hat{\mathbf{V}}$ and with proposal given by algorithm A2.6.
3) Apply algorithm A2.4 to $\hat{S}$ and with proposal distribution (44).
4) Apply algorithm A2.4 to $\hat{\tau}$ and with proposal distribution (44).

Provided that N_1 is large, choose the best predictor among the last N_2 to be the estimated MAP model:

$$\theta_i = \underset{N_1 \le i \le N_2}{\arg\max}\, P(\mathbf{F}_{(r)_i}|X,Y,\tau_i).$$

constrained prediction problems. For all of the 14×20 cases we repeat the experiment 10 times with new $y_i^k = \mathbf{F}^0 X_i + \epsilon$, and $k = 1 \dots 10$. For each experiment we use 400 training points and report the squared reconstruction error on 100 noise free test points and compare the PLS predictor $\hat{\mathbf{F}}^k_{PLS}$ against the MRS predictor $\hat{\mathbf{F}}^k_{MRS}$. To determine the significance of the result we perform the Wilcoxon signed rank test [16] on the 10 mean squared test errors to check if the performance differences are significant (with 5% significance level). We report results on two noise levels $\sigma^2 = 0.1$ and $\sigma^2 = 1$ in figure 2.4.

One can see in figure 2.4 that for most rank constraints and for almost any collinearity condition MRS outperforms PLS. A regime worth noting is when the rank constraint equals the number of input dimension which actually means that there is no rank constraint. In this case, it is known that PLS converges to the standard least squares predictor being optimal for low noise. In our case, we observe that for higher noise levels and if there is no rank constraint PLS overfits (as one would expect from the least squares solution). If the rank constraint is active it leads to possibly underfitting due to the early stopping regularization

Let us pick two rows of the plots in 2.4 for the case that $\sigma^2 = 1$ and analyze in which sense MRS outperforms PLS. In figure 2.5-a we report the relative difference for the rank constraint set to 12. One can see that a significant improvement was obtained consistently for arbitrary levels of collinearity. On the other hand in figure 2.5-b, we show how MRS leads to a predictor with mostly lower performance than PLS if there is no rank constraint (rank = 20), since in this case PLS equals least squares estimation but MRS tries to solve a harder non-convex optimization problem to reach the least squares solution.

[16] Using the matlab signrank implementation.

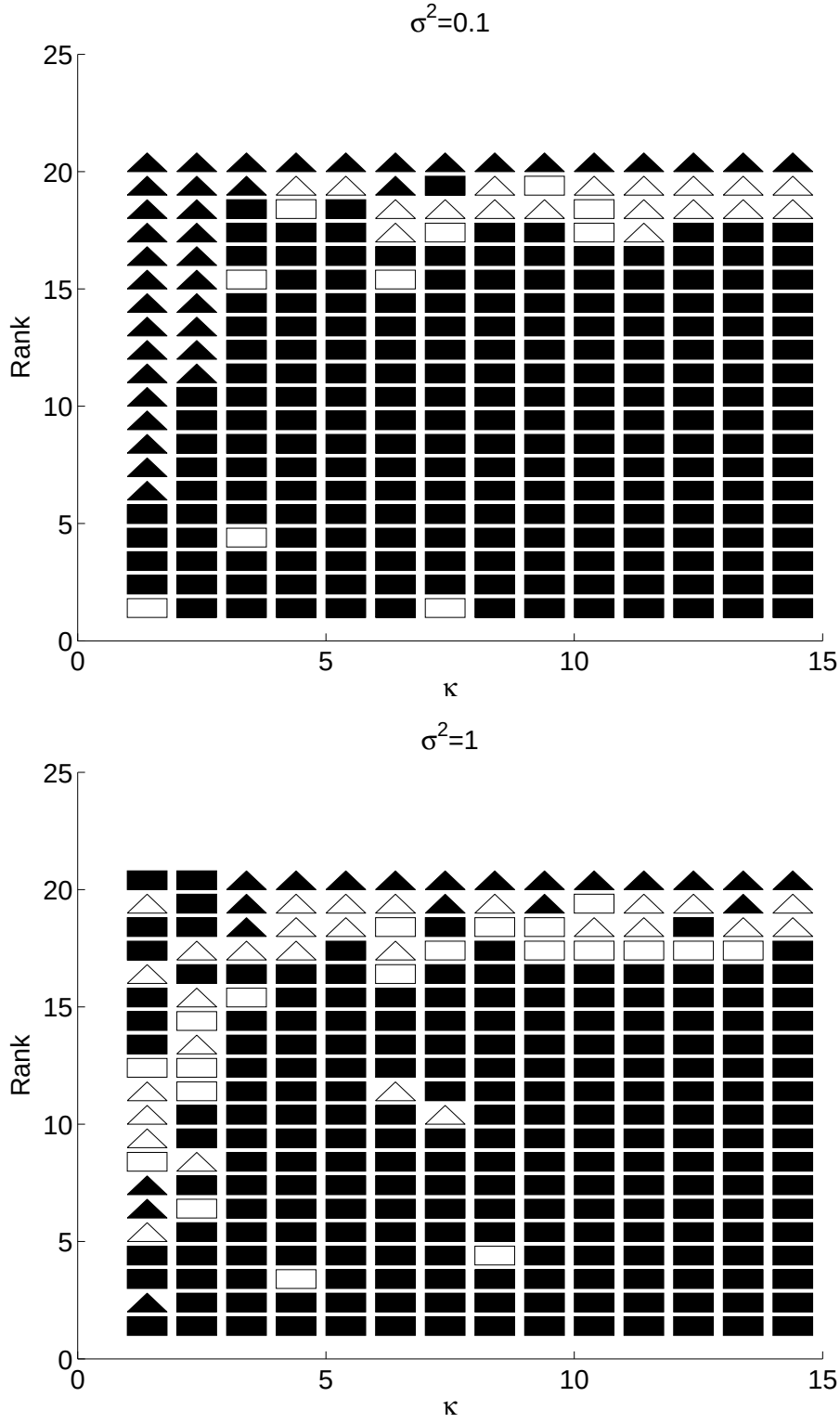
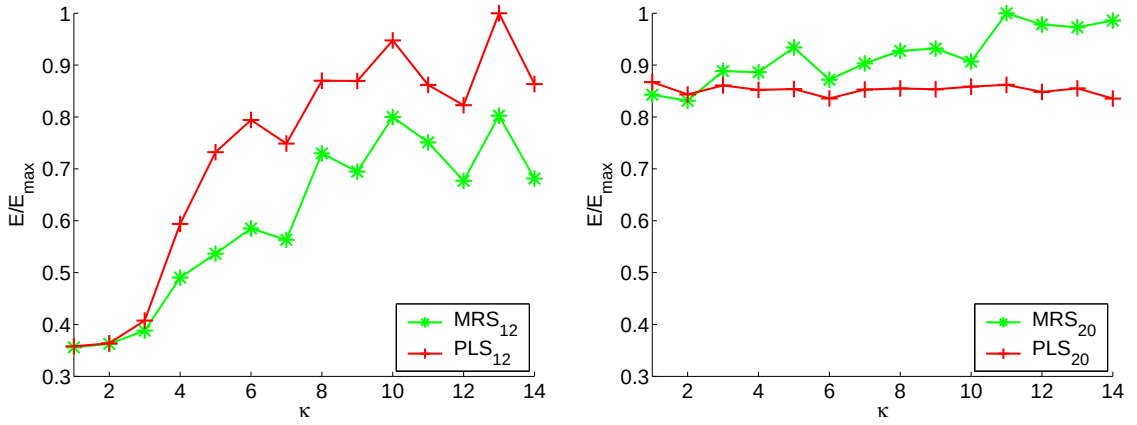


Figure 2.4: Differences between generalization performance of MRS and PLS for various collinearity condition and rank constraint. Triangles indicate that PLS outperforms MRS. Rectangles indicate the opposite. Hollow symbols indicate that no significant difference was observed.



a) b)

Figure 2.5: a) MRS can outperform PLS significantly if there is a rank constraint. b) Since MRS solves a hard non-convex optimization problem, it is not guaranteed to reach the Least Squares Solution.

E denotes the mean squared error, where E_{max} denotes the maximum error over all experiments.

2.6.2 Face denoising: a large-scale problem

In this experiment we apply the *sequential* MRS (S-MRS) to an image denoising, where we draw our data from the face database in [73]. This database contains a training set of 2429 faces and 4548 non-faces, where every face is a 19 x 19 gray-scale image. We perturbed the gray values of each pixel with univariate random noise of magnitude 0.01. The noisy images served as input, and our goal was to reconstruct the original unperturbed image.

To perform denoising, we first applied kernel PCA (see A1.3) to extract 500 nonlinear features from 2400 randomly sampled images, using a Gaussian kernel of width 5. Each input to S-MRS was then given by projecting an image onto these features, yielding a 500 dimensional input space $\mathbb{R}^d$. We set the rank r of the mapping to 50, and trained the algorithm for 50 epochs. In each epoch we train S-MRS with $m = 50$ randomly chosen images (projected onto the 500 dimensional kernel PCA basis). The training outputs were simply the corresponding unperturbed face images of size 19×19.

Denoising performance is shown in Figure 2.6-a. Since our output space is linear and corresponds to 19×19 images, it is possible to visualize the output basis $\mathbf{V}$ by perturbing the mean image in the direction parameterized by each of the column vectors in $\mathbf{V}$ (see Figure 2.6-b). We see in 2.6-a that S-MRS is able to generate a de-noised version of the face despite the strongly corrupted images. Note that the de-noised image sometimes looks like a different person. This effect, is due to kPCA and was explored in [49]. Since MRS and S-MRS optimize the feature matrices $\mathbf{W}, \mathbf{V}$ explicitly during training, we can visualize the directions in $\mathbf{V}$ which live in the space of $R^{19\times 19}$ images. Since $\mathbf{V}$ should span the relevant subspace of the outputs which in our case are faces, one would expect that $\mathbf{V}$ should carry some information about faces. We show in figure 2.6-b that this is indeed the case and that the columns of $\mathbf{V}$ can be directly used. This subspace can be explored, for example, by adding a perturbation to the mean of all faces using the columns of $\mathbf{V}$.

Figure 2.6: a) Denoising performance on 5 face images of the testing set. The original image is on the first row, the noisy image on the second row, and the predicted noise-free image on the third row. b) Illustration of three output feature vectors $\mathbf{v}_i$ in $\mathbf{V}$, one in each row. The significance of these features is shown by perturbing a particular image in the direction parameterized by each of the vectors. The middle column corresponds to the mean image over all faces in the training set, while to the left and right we add $\alpha\,\mathbf{v}_i$ for a small α. The columns correspond respectively to $\alpha \in [-0.2, -0.1, 0, 0.1, 0.2]$.

2.6.3 Probabilistic Inference using the Laplacian Likelihood

In this section, we test our probabilistic rank-constrained regression scheme with a Laplacian likelihood function(corresponding to a non-differentiable L_1 loss function). Our aim is to demonstrate that although the loss function is not differentiable and gradient descent procedures are not applicable, sampling-based schemes can be applied successfully. To this end, we create a synthetic regression problem with two-dimensional inputs and two-dimensional outputs and look for the best predictor with rank one. We create random input data as in the synthetic experiment in 2.6.1 above where we scale the second input dimension by 0.2 to obtain collinear inputs. We use a random transformation matrix to obtain the outputs, which we perturb by Gaussian noise with variance 0.1. The training and the test set contain 500 points each. From the training set we perturb two randomly selected vectors y_a, y_b by $\hat{y}_a = y_a + t \cdot v$, and $\hat{y}_a = y_a - t \cdot v$ where v is the minor principal component of the outputs. Thus we introduce outliers in a controlled manner. For the burn-in phase of the sampling-based algorithm using the Laplacian likelihood we set N_1, N_2 equal to 500. We consider the best predictor of the last 500 steps.

We show in figure 2.7-a) the output data with both outliers y_a, y_b for two states $t = 0$, $t = 1$ of the experiment. In figure 2.7-b) one can see that standard L_2 loss function and PLS is going to fit more and more the two outliers whereas the MAP estimate using the Laplacian likelihood function remains almost unperturbed. Thus, clearly using sampling we can use the likelihood function which is right for the task at hand and do not need to constrain ourselves on differentiable loss functions.

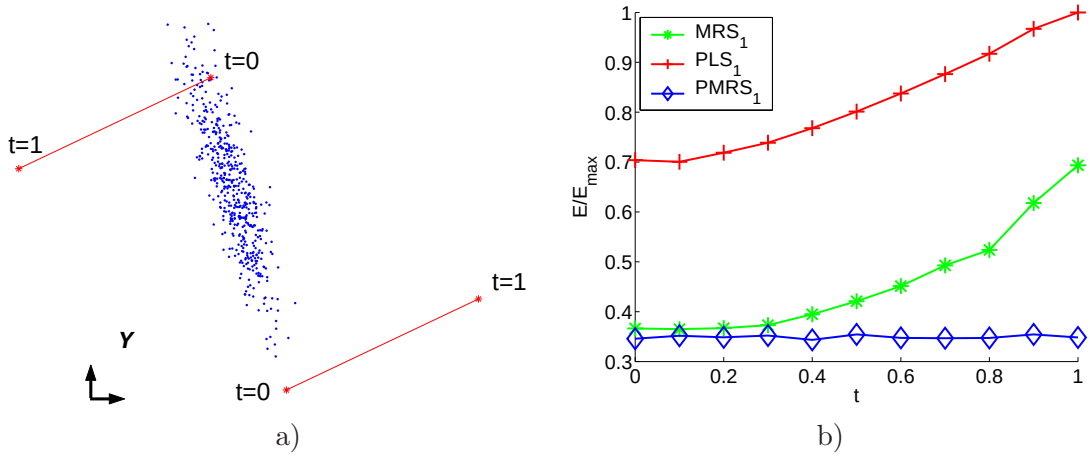


Figure 2.7: Sensitivity of the MAP Estimator using the Laplacian likelihood function in contrast to the PLS and L_2 minimization.(We used the maximum error for normalization.)

2.7 Conclusion

We have introduced a new framework, MRS, for regression between multi-dimensional spaces. Our method tries to find the optimal input and output subspace for the given task at hand using a gradient descent procedure on the manifold of orthogonal matrices. Note that, for example in KDE the regression problem is as high-dimensional as many training examples one have (see section 1.3). Thus, in this situations the restriction on subspaces is a reasonable regularization mechanism. In contrast to other existing heuristics like PLS or PCR which also choose subspaces for regression, our introduced method is derived from a pure optimization viewpoint, trying to minimize single objective function. This allows us to extend classical reduced rank regression with additional regularizers and adapt it to arbitrary differentiable loss functions. A major disadvantage of the gradient-based technique is that it depends on a starting point since a rank constraint leads to a non-convex optimization problem. However, solutions from classical techniques can be used as a reasonable starting point for optimization. For the case of non differentiable loss functions we proposed the maximum a-posteriori solution which can be found by MCMC sampling. We have investigated MCMC sampling for our model class and discussed how one can sample from Stiefel manifolds. Using general sampling strategies allows us to pick *any* suitable loss function for the task at hand, not necessarily constrained to be differentiable.

Chapter 3

Strategies For Continuous and Discrete Pre-Image Problems

In the middle of difficulty lies opportunity.
– Albert Einstein –

In this chapter we are concerned with the problem of reconstructing patterns from their coordinate representation in feature space. We will introduce algorithms which attempt to reconstruct possibly structured patterns from a feature space embedding.

3.1 Introduction

In this chapter we are concerned with the pre-image problem. Besides being one of the cornerstones in the generalized inference setting of KDE, calculating pre-images is of wide interest in kernel methods. Applications of pre-image techniques cover for example *reduced set methods* in [12], *denoising* in [61] and *hyper-resolution* using kernels in [49]. Let us give a definition of the pre-image problem first:

Problem P3.1 The Pre-Image Problem:

Given a point $\mathbf{\Psi}$ *in a feature space* $\mathcal{F}_z$*, find a pattern* z^* *in the set* $\mathcal{Z}$ *such that the feature map* ϕ_z *maps* z^* *to* $\mathbf{\Psi}$*, i.e.:*

$$\mathbf{\Psi} = \phi_z(z^*). \tag{1}$$

A special case of the problem P3.1 is the case when $\mathbf{\Psi}$ itself is given by a linear expansion of patterns:

Problem P3.2 The Pre-Image Problem for Expansions:

Given a point $\mathbf{\Psi}$ *in a feature space* $\mathcal{F}_z$ *expressed by a linear combination of patterns* $\{x_1, \ldots, x_N\} \subset \mathcal{Z}^N$*, i.e.:*

$$\mathbf{\Psi} = \sum_{i=1}^{N} \alpha_i \phi_z(x_i),$$

find a pattern z^* *in space* $\mathcal{Z}$ *such that*

$$\mathbf{\Psi} = \sum_{i=1}^{N} \alpha_i \phi_z(x_i) = \phi_z(z^*).$$

Obviously solving pre-image problems corresponds to an inverse problem, since ideally $z^* = \phi_z^{-1}(\mathbf{\Psi})$ and thus solving a pre-image problem requires to invert the feature map. Unfortunately, in the most common cases it turns out that the problem of inverting ϕ belongs to the class of *ill-posed* problems. Let us first review the properties of well and ill-posed problems.

Definition D3.1 Well-Posedness of Inversion Problems, see [90]: *The problem*

$$A(x) \quad = \quad y_0, \tag{2}$$

*where $A : D(A) \subset \mathcal{X} \rightarrow \mathcal{Y}$ is an operator, $y_0 \in \mathcal{Y}$ is the target, and $x \in D(A)$ is an solution to the operator equation (2), is said to be **well posed** in the Hadamard sense, if it satisfies the following three conditions:*

1. *Existence. For every target $y \in \mathcal{Y}$, there exists a solution $x \in D(A) \subset \mathcal{X}$ with $y = A(x)$.*
2. *Uniqueness. For any pair of inputs $x, z \in D(A) \subset \mathcal{X}$, it follows that $A(x) = A(z)$ if and only if $x = z$.*
3. *Continuity (stability). The mapping is continuous, that is, for any $\epsilon > 0$ there exists δ such that the condition $||x - z||_{\mathcal{X}} < \delta \Rightarrow ||A(x) - A(z)||_{\mathcal{Y}} < \epsilon$, with $|| \cdot ||_{\mathcal{Y}}$ representing an appropriate distance metric.*

*The problem is said to be **ill posed**, if any of these three conditions are not satisfied. If A is a linear mapping we speak about linear ill-posed problems and of non-linear ill-posed problems otherwise.*

Roughly speaking Hadamard defined an inverse problem as in equation (2) as *ill posed* if the solution is not unique, does not exist or if it is not a continuous function of the data – i.e. if a small perturbation of the data can cause an arbitrarily large perturbation of the solution.

We could use definition D3.1 to categorize kernels k and their induced feature maps ϕ_k according to their inversion properties, however it turns out that the most critical property is existence and thus we will focus our discussion on the existence property.

3.1.1 The pre-image problem: An ill-posed problem.

Let us consider first the case that $\mathbf{\Psi}$ is given by an expansion. One can show that if the kernel can be written as $k(x, x') = f_k(x^\top x')$ with an invertible function f_k (e.g., polynomial kernels with $k(x, x') = (x^\top x')^d$ with odd d), then one can compute the pre-image analytically as

$$\boldsymbol{z} = \sum_{i=1}^{m} f_k^{-1} \left(\sum_{j=1}^{N} \alpha_j k(x_j, \boldsymbol{e}_i) \right) \boldsymbol{e}_i,$$

where $\{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_m\}$ is any orthonormal basis of input space.

Unfortunately this is rather the exception than the usual case. One reason is that several patterns in input space can have the same feature space representation. In particular kernels $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ which are invariant under some transformations $\mathcal{I}(t) : \mathcal{X} \times \mathbb{R} \rightarrow \mathcal{X}$, i.e.: $k(x_i, x_j) = k(I(t, x_i), I(t, x_j))$, induce non injective feature maps and thus do not fulfill the uniqueness criteria. For a discussion of invariant kernels, see e.g. [78, chapter 11].

Another reason is that the span of the data points does not need to coincide with the image of the feature map ϕ_k. Even worse, for the most widely used radial basis function kernel *any* non-trivial linear combination of mapped points will leave the image of ϕ_k and this guarantees that there does not exist any pre-image. Therefore kernels which are not invertible and/or have built-in invariance properties imply ill-posed pre-image problems because they do not fulfill the existence and the uniqueness criteria of D3.1.

An additional difficulty arises in the pre-image setting of KDE where $\mathbf{\Psi}$ is an estimated point represented by estimated kPCA coordinates (see chapter 1). In KDE it can happen that the estimated representation $T_{\mathcal{F}}\phi_k(x^*)$ is not part of the image of the output feature map ϕ_l, where $T_{\mathcal{F}}$ is the learned map among input and output feature spaces. This is likely since there is no constraint in the KDE regression problem formulation that forces predictions $\mathbf{\Psi} = T_{\mathcal{F}}\phi_k(x^*)$ to be reconstructable. [1]

Summarizing we see that inverting the feature map is likely to be an ill-posed problem. Thus to overcome the ill-posed formulation [12] suggested to relax the original pre-image problem.

3.1.2 Relaxation of the pre-image problem

A straightforward relaxation of the inversion problem is to drop the requirement on the pattern z^* to be the exact pre-image but to be as close as possible to an ideal pre-image. Thus we consider the following approximative pre-image problem:

Problem P3.3 The Approximate Pre-Image Problem:

Given a point $\mathbf{\Psi}$ in $\mathcal{F}_z$, find a corresponding pattern $z^* \in \mathcal{Z}$ such that

$$z^* = \arg\min_{x \in \mathcal{Z}} ||\phi_z(x) - \mathbf{\Psi}||_2^2. \tag{3}$$

We will denote henceforth the objective function by $\mathcal{L} : \mathcal{Z} \to \mathbb{R}$.

Note that the problem P3.3 is an optimization problem and not an inversion problem, and that in the case where the inverse map ϕ^{-1} exists, $\phi^{-1}(\mathbf{\Psi})$ equals the optimal solution z^* of (3) in P3.3. Otherwise the found z^* is the solution with minimal Euclidean defect $||\phi_z(z^*) - \mathbf{\Psi}||_2^2$. Furthermore note that the new problem is formulated

- in a high and possibly infinite dimensional feature space (illustrated in figure 3.1),
- and the optimization is over an arbitrary set in the input space.

It is not straightforward to see that a tractable algorithm solving problem P3.3 should exist. Fortunately for some interesting sets $\mathcal{Z}$ we can formulate algorithms which can be used to solve P3.3 and this is the subject of this chapter. These algorithms are not applicable in general cases for $\mathcal{Z}$, but are always going to be *tailored* specifically to $\mathcal{Z}$ by using special properties inherent to the set $\mathcal{Z}$.

We begin in section 3.2 with the case that $\mathcal{Z}$ is a subset of a real vector space $\mathbb{R}^d$. We will review existing pre-image techniques which are only applicable in this case and propose a new pre-image algorithm based on learning. Furthermore we will focus on feature mappings which are induced by a Gaussian kernel, since it is the most widely used kernel in practice. In section 3.3 we will compare the introduced algorithms on the problem of visualizing eigenvectors obtained by kPCA.

[1] One can consider a modification of the regression formulation used in KDE such that predictions lie on an a-priori defined manifold in the feature space. This would lead to a regression task with *structured* outputs and is subject of current research.

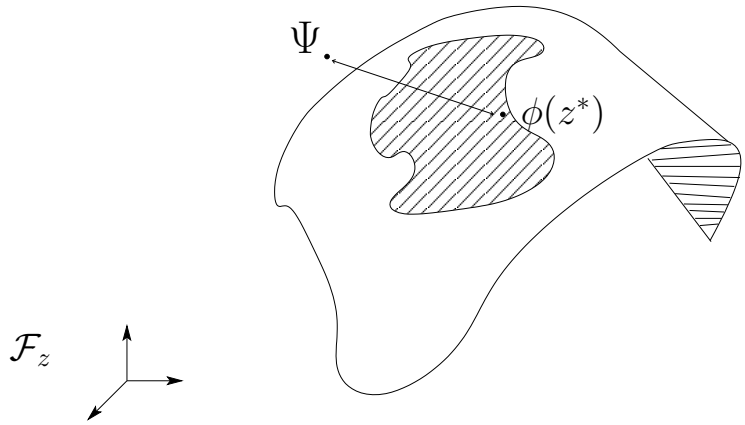


Figure 3.1: The approximate pre-image problem: Searching for a pre-image z^* of a point $\mathbf{\Psi}$ given in feature space $\mathcal{F}_z$ corresponds to finding the nearest point $\phi(z^*)$ on a nonlinear manifold.

In the succeeding section 3.4 we introduce pre-image strategies for the case that $\mathcal{X}$ is not a subset of a real vector space but a discrete set with some structural information. Classical algorithms wont be applicable in this case, since they are based on smooth optimization. We introduce pre-image algorithms for sequences and labeled graphs and investigate in section 3.5 the application of calculating the *mean* of sequences.

3.2 Pre-Images by smooth optimization

In this section we review several existing approaches to solve the *approximate* pre-image problem and propose new techniques. We will start with pure optimization-based approaches and continue with strategies based on learning which can take advantage of additional prior information on the pre-image.

3.2.1 Gradient Descent

The most straightforward approach is to use non-linear optimization techniques to solve the approximate pre-image problem. In the case that Ψ is given by an expansion $\sum_{i=1}^{N} \alpha_i \phi_k(x_i)$, one can express the objective function in (3) directly by using the kernel trick itself:

$$\mathcal{L}(x) = ||\phi_k(x) - \mathbf{\Psi}||_2^2 = k(x,x) - 2\sum_{i=1}^{N} \alpha_i k(x_i, x) + C, \tag{4}$$

where C denotes some constant independent of x. Taking the derivatives of (4) is straightforward as long as the kernel function is smooth and continuous in its argument.

In contrast, in the setting of KDE where one has given an output kernel k_l, the point $\mathbf{\Psi}$ is specified by its r predicted kPCA coordinates $\boldsymbol{V}_r^\top \boldsymbol{y}^* = [\beta^1, \ldots, \beta^r] = T\phi(x^*)$ only (seeA1.3). Thus the objective function takes the form:

$$\mathcal{L}(x) = ||\boldsymbol{V}_r^\top \boldsymbol{y}^* - \phi_l(y)||_2^2 = k_l(y,y) - 2\sum_{i=1}^{r} \beta^i \sum_{j=1}^{N} v_r^j k_l(y_j, y) + C, \tag{5}$$

where

$$\boldsymbol{v_s}^\top \phi_l(y) = \left(\sum_j v_s^j \phi_l(y_j) \right)^\top \phi_l(y) = \sum_j v_s^j k_l(y_j, y)$$

denotes the projection of $\phi_l(y)$ to the s-th. eigenvector of the kPCA basis $\boldsymbol{V}_r$. One can still formulate gradients, however evaluating the gradient is expensive since one needs to evaluate all kPCA projections which requires to evaluate the kernel function on the whole training data for each N principal directions. Nevertheless, a similar strategy was pursued in by [61] who applied pre-image techniques for image de-noising.

The problem with a plain gradient descent approach is that equation (4) and (5) are mostly not linear and not convex. For this reason one has to use multiple starting values to initialize the gradient descent in the hope that one of the found minima is good enough. People therefore considered to speed-up pre-image techniques, for example by taking advantage of special properties of the used kernel function. One such example is the fixed-point iteration method introduced in [78, chapter 18] which we discuss next.

3.2.2 The Fixed-Point Iteration Method

For kernels with the property $k(z,z) = 1$, e.g. the Gaussian kernel, one can reformulate the problem $P3.3$ as follows:

$$z^* = \arg\min_{x \in \mathcal{Z}} ||\phi_z(x) - \boldsymbol{\Psi}||_2^2 = \arg\max_{x \in \mathcal{Z}} \ \phi_z(x)^\top \boldsymbol{\Psi}. \tag{6}$$

The reformulation is possible since all pre-images have the same distance to the origin [2], and therefore the dot-product contains all relevant information. Now for example for kernels of the form $k(x,z) = k(||x-z||^2)$ and the case that $\boldsymbol{\Psi}$ is given as expansion it is possible to derive an efficient fixed-point algorithm as follows:

$$\begin{aligned} 0 &= \frac{\partial}{\partial z}\phi_k(z)^\top \boldsymbol{\Psi} \\ &= \sum_{i=1}^N \alpha_i \frac{\partial}{\partial z}\phi_k(x_i)^\top \phi_k(z) \\ &= \sum_{i=1}^N \alpha_i k'(x_i, z)(x_i - z) \end{aligned}$$

which leads to the relation

$$z = \frac{\sum_{i=1}^N \alpha_i k'(x_i,z) x_i}{\sum_{i=1}^N \alpha_i k'(x_i,z)}.$$

This identity does not hold for an arbitrary z_0 but only for the solution z^*. The idea is now to construct a sequence z_1, z_2 via

$$z_{t+1} = \frac{\sum_{i=1}^N \alpha_i k'(x_i,z_t) x_i}{\sum_{i=1}^N \alpha_i k'(x_i,z_t)}, \tag{7}$$

such that ideally z_t converges to the fixed-point in (7). In the same manner one can obtain a fixed-point based iteration scheme for the KDE setting. Using the same notation as in section 3.2.1 we obtain

$$z_{t+1} = \frac{\sum_{i=1}^r \beta^i \sum_{j=1}^N v_j^i k'(x_j,z_t) x_j}{\sum_{i=1}^r \beta^i \sum_{j=1}^N v_j^i k'(x_j,z_t)}. \tag{8}$$

Since one does not need to perform any line-search which is typically needed in gradient descent methods, fixed-point based techniques are generally faster than gradient descent

[2] $||\phi_k(z)|| = \sqrt{k(z,z)} = 1$ by construction.

methods. However, the method suffers similar to gradient descent techniques from evaluating all kernel functions for each optimization step, though it needs overall much less evaluations than plain gradient descent does.

Unfortunately, the fixed-point technique tends to be numerically instable in practice if initialized randomly. This phenomenon might be due to a) the fact that the denominator can get too small for an arbitrary chosen start point z_0 during the iteration, and b) there exist several regions of attraction which can influence each other and thus might lead to oscillation or even divergence. A common recipe in this case is to restart the iteration with different starting points z_0 often and choose the best candidate of the converged runs. We present the fixed-point algorithm in A3.1.

An interesting fact about the fixed-point method is that the pre-image obtained is in the span of the data points x_i. This is a major difference to a pure optimization based technique like gradient descent where the full input space is explored. Thus, in the fixed-point method the pre-image is constructed using data points explicitly. For example, the fixed-point algorithm can never generate a pre-image which is orthogonal to the span of the training data, in contrast to the gradient descent method. Nevertheless, the fixed-point algorithm shows that in principle one can relate the pre-image not only to a single point $\mathbf{\Psi}$ in feature space but to a complete set of points D_N and thus use side-information provided by these additional data. In the next section, we discuss a classical technique from statistics which is used for pre-image calculation and which is entirely based on such a strategy.

3.2.3 Multi-Dimensional Scaling based technique

Multi-Dimensional Scaling (MDS) deals in particular with the problem of finding a lower dimensional representation $x_1, \ldots, x_N \in \mathbb{R}^{d_2}$ of a set of points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^{d_1}$ where $d_1 \gg d_2$ and ideally the distances are preserved, i.e.:

$$d_{\mathbb{R}^{d_1}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = d_{\mathbb{R}^{d_2}}(x_i, x_j)$$

Often, such a set of points does not exist and one solves the optimization problem:

$$\{x_1, \ldots, x_N\} = \underset{x_1,\ldots,x_N}{\arg\min} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(d_{\mathbb{R}^{d_1}}(\boldsymbol{x}_i, \boldsymbol{x}_j) - d_{\mathbb{R}^{d_2}}(x_i, x_j)\right)^2 .$$

In contrast to the standard MDS setting, in pre-image computation we are just interested in finding the lower dimensional representation of a single point namely the pre-image z^* only. The basic idea behind the MDS method for pre-image computation is to declare that the distance from the point $\mathbf{\Psi}$ to each point $\boldsymbol{x}_i$ of a reference set $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ should be related to the distance of the pre-image z^* to the pre-images $x_1, \ldots, x_N$ of the reference set. This is insofar a reasonable assumption since the feature map is usually considered to be smooth. Obviously, these distances will not be the same and we need to obtain the input distance from the feature map. In the following we will describe the necessary steps to obtain the input distances for the Gaussian and Polynomial kernel and we will discuss how a pre-image can be obtained using least squares minimization leading to a closed form solution of the optimal approximate pre-image.

Let us consider the used distance for the Gaussian kernel $k(x, y) = e^{-\gamma||x-y||^2} = e^{-\gamma d_{\mathcal{Z}}^2(x,y)}$:

$$d_{\mathcal{Z}}(x, y) = \sqrt{-\frac{\log k(x, y)}{\gamma}}.$$

Algorithm A3.1 FIXED-POINT PRE-IMAGE CALCULATION FOR KDE

Given data $D_N = \{x_i\}_{i=1}^N$, a feature map ϕ_k from a kernel k, an orthogonal basis $\boldsymbol{V}_r = [\boldsymbol{v}^1, \ldots, \boldsymbol{v}^r]$ of the span of points D_N in the feature space and the number of desired restarts n. Let the predicted r coordinates of the point $\boldsymbol{y}^* = T_{\mathcal{F}}\phi(x^*)$ according to the orthogonal basis $\boldsymbol{V}_r$ in the feature space $\mathcal{F}_k$ be denoted as $[\beta_1, \ldots, \beta_r]$.

1. Choose randomly an initial value z_0 from D_N.
2. Calculate
$$z_{t+1} = \frac{\sum_{i=1}^r \beta^i \sum_{j=1}^N v_j^i k'(x_j, z_t) x_j}{\sum_{i=1}^r \beta^i \sum_{j=1}^N \boldsymbol{v}_j^i k'(x_j, z_t)},$$
until convergence.

   If the denominator is smaller than some ϵ, restart at 1.

Repeat n times the pre-image computation and take the best z from n trials according to the lowest distance in feature space $||[\beta_1, \ldots, \beta_r] - \boldsymbol{V}_r^\top \phi_k(z)||$.

Given the distance among two points in feature space

$$d_{\mathcal{F}_k}(\boldsymbol{x}, \boldsymbol{y})^2 = ||\phi(x) - \phi(y)||^2 = 2 - 2k(x, y),$$

it is possible to obtain the squared input distance as

$$d_{\mathcal{Z}}(x, y)^2 = -\frac{1}{\gamma} \log\left(1 - \frac{1}{2} d_{\mathcal{F}_k}(x, y)^2\right). \tag{9}$$

Thus we see that, given just the expression of the kernel function we can retrieve the distance of two points in input space. Now, lets consider the Euclidean distance in feature space. As we noted above, the distance among two points $\boldsymbol{x}$ and $\boldsymbol{\Psi}$ can be calculated without their input representation. In the case that $\boldsymbol{\Psi}$ is given as expansion, the Euclidean distance between $\boldsymbol{\Psi}$ and a point $\boldsymbol{x}$ in feature space is given by

$$d_{\mathcal{F}_k}(\boldsymbol{x}, \boldsymbol{\Psi})^2 = k(x, x) - 2\sum_{i=1}^N k(x_i, x) + \sum_{i=1}^N \sum_{j=1}^N \alpha_i, \alpha_j k(x_i, x_j).$$

Even simpler, if $\boldsymbol{\Psi}$ is given by its coordinates $[\beta_1, \ldots, \beta_r]$ with respect to an orthogonal basis $\boldsymbol{V}_r$ only, then the feature space distance is simply the Euclidean distance among coordinates, i.e..:

$$d_{\mathcal{F}_k}(\boldsymbol{x}, \boldsymbol{\Psi})^2 = ||\boldsymbol{V}_r^\top \phi_k(x) - \boldsymbol{V}_r^\top \boldsymbol{\Psi}||^2 = \sum_{i=1}^r (\gamma_i - \beta_i)^2, \tag{10}$$

where $\gamma = \boldsymbol{V}_r^\top \phi_k(x)$ are the r coordinates of $\phi_k(x)$ respective $\boldsymbol{V}_r$.

Now calculating the input space distances between the point of interest $\boldsymbol{\Psi}$ and each reference point in some training data D_N via (9), we obtain a distance vector $d^2 := [d_1^2, \ldots, d_N^2]$ of size N. This vector can be used to pose an optimization problem similar to the original MDS optimization problem

$$z^* = \arg\min_{z \in \mathcal{Z}} \sum_{i=1}^N (d_{\mathcal{Z}}^2(z, x_i) - d_i^2)^2.$$

Unfortunately this is as hard to optimize as optimizing the distance in feature space directly. However considering the term $d^2_{\mathcal{Z}}(z, x_i) - d_i^2$ we see that

$$d^2_{\mathcal{Z}}(z^*, x_i) - d_i^2 = x_i^\top x_i + z^{*\top} z^* - 2x_i^\top z^* - d_i^2,$$

and thus for the ideal case $d^2_{\mathcal{Z}}(z^*, x_i) - d_i^2 = 0$ we obtain the identities

$$2x_i^\top z^* = x_i^\top x_i + z^{*\top} z^* - d_i^2, \text{ for all } 1 \leq i \leq N. \tag{11}$$

Let us write all N equations in matrix form as

$$2X^\top z^* = d_0^2 - d^2 + \mathbf{1}_N z^{*\top} z^* \tag{12}$$

where $d_0^2 = [x_1^\top x_1, \ldots, x_N^\top x_N] \in \mathbb{R}^N$ and $X = [x_1, \ldots, x_N] \in \mathbb{R}^{d \times N}$, and $\mathbf{1}_N = [1, \ldots, 1] \in \mathbb{R}^d$. It seems that we did not gain anything since the desired and unknown pre-image z^* appears also on the right side. But note that from (11) and the definition of d_0 we can get yet another relationship

$$z^{*\top} z^* = 2x_i^\top z^* + d_i^2 - d_{0_i}^2, \text{ for all } 1 \leq i \leq N. \tag{13}$$

If we average both sides of (13) over all used N points and require that the points in D_N are centered, that is $\sum_{i=1}^N x_i = 0$, we can rewrite the unknown quantity $z^{*\top} z^*$ as

$$\begin{aligned} z^{*\top} z^* &= \frac{1}{N} \sum_{i=1}^N 2x_i^\top z^* + d_i^2 - d_{0_i}^2 \\ &= \frac{1}{N} \sum_{i=1}^N \left(d_i^2 - d_{0_i}^2\right) + 2\frac{1}{N} \underbrace{\left(\sum_{i=1}^N x_i\right)^\top}_{=0} z^* \\ &= \frac{1}{N} \sum_{i=1}^N \left(d_i^2 - d_{0_i}^2\right) \end{aligned}$$

and we obtain the new equation

$$2X^\top z^* = d_0^2 - d^2 + \frac{1}{N} \mathbf{1}_N \mathbf{1}_N^\top \left(d^2 - d_0^2\right). \tag{14}$$

The pre-image z^* can now be obtained by the least squares solution of (14)

$$z^* = \frac{1}{2}(XX^\top)^\dagger X \left(d_0^2 - d^2\right) + \frac{1}{N}(XX^\top)^\dagger X \mathbf{1}_N \mathbf{1}_N^\top \left(d^2 - d_0^2\right),$$

where $(XX^\top)^\dagger$ denotes the pseudo-inverse of $(XX^\top)$. Note that this can be simplified further because we required the N data points in D_N to be centered. Thus the second term $\frac{1}{2}(XX^\top)^{-1} X \mathbf{1}_N \mathbf{1}_N^\top \left(d^2 - d_0^2\right)$ equals zero since $X\mathbf{1}_N = \sum_{i=1}^N x_i = 0$ by construction. Therefore the optimal pre-image reconstruction is given by

$$z^* = \frac{1}{2}(XX^\top)^\dagger X \left(d_0^2 - d^2\right). \tag{15}$$

Note that the nature of the kernel function entered only in the calculation of the input distance vector in d^2, thus in a similar manner one can get pre-images for any kernel by

replacing the calculation in (9). For example for the polynomial kernel $k(x_1, x_2) = (x_1^\top x_2)^q$ we would have

$$d_{\mathcal{Z}}^2(x, y) = \sqrt[q]{k(x, x)} + \sqrt[q]{k(y, y)} - 2\sqrt[q]{k(x, y)}. \tag{16}$$

The MDS based pre-image calculation was pioneered by [54], where as their formulation require to calculate a singular value decomposition of the data beforehand. This is indeed not necessary and our introduced version is easier and faster.

MDS uses the distances to some other points as side-information to reconstruct the pre-image. Note that it is non-iterative in contrast to fixed-point methods but still needs to solve a least squares problem for each prediction. MDS based pre-image calculation is in particular well suited for KDE because it only requires calculation of distances which equal the Euclidean distance between coordinates, see (10). If one stores a-priori the kPCA coordinates of all training points one does not require any output kernel function evaluation during testing which saves time. Furthermore Kwok suggested to use only n nearest neighbors instead of all N points which gives a further speedup. We give a summary of the MDS-based pre-image algorithm in A3.2.

3.2.4 Pre-Images by Learning

Unfortunately, all techniques introduced so far suffer from one of the following points. They are

- formulated as a difficult nonlinear optimization problem with local minima requiring restarts and other numerical issues.
- being computationally inefficient, given that the problem is solved individually for each new point.
- not being the optimal approach depending on the task (e.g., we may be interested in minimizing a classification error rather then a distance in feature space.)

Consider the case of KDE where we are given training data to learn a map $t_{\mathcal{Z}}$ between the sets $\mathcal{X}$ and $\mathcal{Y}$. Using pre-image techniques presented so far, our prediction map Γ consists of a possibly non-convex optimization on each new presented point x^*, since we have to look for a suitable $y^* \in \mathcal{Y}$ given its predicted coordinates. This is clearly a disadvantage since it does not allow to use KDE in more time-critical scenarios. A further aspect shared by all the methods discussed so far is that they do not explicitly make use of the fact that we have *labeled examples* of the unknown pre-image map: specifically, if we consider any point in $x \in \mathcal{Z}$, we know that the pre-image of $\phi_k(x)$ is simply x. It may not be the only pre-image, but this does not matter as long as it minimizes the value of P3.3 (see page 47).

We propose a method which makes heavy use of this information and can resolve all the mentioned difficulties: The simple idea is to estimate a function Γ by *learning* the map $\boldsymbol{\Psi} \to z^*$ from examples $(\phi_k(z^*), z^*)$. Depending on the learning technique used this means: After estimating the map Γ, each use of this map can be computed very efficiently since pre-image computation equals evaluating a (possibly non-linear) function. Thus, there are no longer issues with complex optimization code during testing phase. Note that this approach is unusual in that it is possible to produce an *infinite* amount of training data (and thus expect to get good performance) by generating points in $\mathcal{Z}$ and labeling them using the identity $\phi_k(z) = z$. Furthermore, it is often the case that we have knowledge about the distribution over the possible pre-images, e.g., when de-noising

Algorithm A3.2 PRE-IMAGE CALCULATION WITH MULTI DIMENSIONAL SCALING

Given data $D_N = \{x_i\}_{i=1}^N$, a feature map ϕ_k from a kernel k and coordinate matrix $R_N = [\gamma_1, \ldots, \gamma_N] \in \mathbb{R}^{r \times N}$ of the data D_N respective some orthogonal basis $\boldsymbol{V}_r$. Furthermore let n be the number of used nearest neighbors. Let the predicted r coordinates of the point $\boldsymbol{y}^* = T_{\mathcal{F}}\phi(x^*)$ according to the orthogonal basis $\boldsymbol{V}_r$ in the feature space $\mathcal{F}_k$ be denoted as $[\beta_1, \ldots, \beta_r]$.

1. Choose the n nearest neighbor coordinates $\{\gamma^{i_1}, \ldots, \gamma^{i_n}\}$ of $[\beta_1, \ldots, \beta_r]$.
2. Center the nearest neighbors via $\bar{x} = \frac{1}{n}\sum_i^n x_i,\ x_i = x_i - \bar{x}$.
   All further calculation requires $\sum_i x_i \overset{!}{=} 0$.
3. Calculate the Euclidean distances $d_{\mathcal{F}}(x_i, \boldsymbol{y}^*)$ using (10).
4. Calculate the squared input space distance vector $d^2 = [d_1^2, \ldots, d_n^2]$ according to (9) or (16).
5. Calculate the matrix $XX^\top$ of the nearest neighbors $X = [x_{i_1}, \ldots, x_{i_n}]$ and the squared input space norm vector $d_0 = \operatorname{diag}(XX^\top)$.

The optimal pre-image z^* is given by $z^* = \frac{1}{2}(XX^\top)^\dagger X(d_0^2 - d^2) + \bar{x}$.

digits with kPCA, one expects as a pre-image something that looks like a digit, and an estimate of this distribution is actually given by the original data. Taking this distribution into account, it is conceivable that a learning method could outperform the naive method, that of P3.3, by producing pre-images that are subjectively preferable to the minimizers of P3.3. In the following we consider the training data $\{\boldsymbol{x}_i\}_{i=1}^N$ that we are given in our original KDE learning problem as the training data for pre-image learning.

Details of the learning approach

We seek to estimate a function $\Gamma : \mathcal{F}_k \to \mathcal{Z}$ with the property that, at least approximately,

$$\Gamma(\phi_k(x_i)) \approx x_i, \quad 1 \leq i \leq N.$$

If we were to use regression using the kernel k corresponding to $\mathcal{F}_k$, then we would simply look for weight vectors $\boldsymbol{w}_j \in \mathcal{F}_k$, $j = 1, \ldots, d$ such that $\Gamma_j(\boldsymbol{\Psi}) = \boldsymbol{w}_j^\top \boldsymbol{\Psi}$, and use the kernel trick to evaluate Γ. However, in general we may want to use a kernel κ which is different from k, and thus we cannot perform our computations implicitly by the use of a kernel. Fortunately, we can use the fact that although the mapped data may live in an infinite-dimensional space $\mathcal{F}_k$, any finite data set spans a subspace of finite dimension. A convenient way of working in that subspace is to choose a basis $\boldsymbol{V}_r$ and to work in coordinates. This basis could be, for instance, obtained by kPCA.

Let $R_N = [\gamma^1, \ldots, \gamma^N] \in \mathbb{R}^{r \times N}$ denote the coordinate matrix of the data D_N with regards to the orthogonal basis $\boldsymbol{V}_r$. Given $\boldsymbol{V}_r$, we can decompose Γ into the concatenation of $\hat{\Gamma} : \mathbb{R}^r \to \mathbb{R}^d$ and $\boldsymbol{V}_r^\top$, such that

$$z^* = \Gamma(\boldsymbol{\Psi}) = \hat{\Gamma}(\boldsymbol{V}_r^\top \boldsymbol{\Psi}).$$

Note that the task is now to find the pre-image map $\hat{\Gamma}_j : \mathbb{R}^r \to \mathbb{R}^j$, $j = 1, \ldots, \dim \mathcal{Z}$ from feature space coordinates $\boldsymbol{V}_r^\top \boldsymbol{\Psi} \in \mathbb{R}^r$ to the input space $\mathbb{R}^{\dim \mathcal{Z}}$. Note this is a standard regression problem for the N training points $\boldsymbol{x}_i$. Thus we can use any arbitrary regression method, e.g. kernel ridge regression with a kernel κ different from kernel k. The map $\hat{\Gamma}_j$ would be then completely given once the ridge regression weights $\{\beta^1, \ldots, \beta^{\dim \mathcal{Z}}\} \in \mathbb{R}^{N \times \dim \mathcal{Z}}$ are specified. To find the regression coefficients in ridge regression we would

have to solve the following optimization problem (for a discussion on ridge regression, see also chapter 1)

$$\boldsymbol{\beta}^j = \arg\min_{\boldsymbol{\beta}^j} \sum_{i=1}^{m} \mathsf{L}\left(x_i^j, \Gamma_j(x_i|\boldsymbol{\beta}^j)\right) + \lambda||\boldsymbol{\beta}^j||^2.$$

Here, L denotes some loss function used for pre-image learning and λ denotes the ridge parameter. Let us give a detailed example.

Example E3.1 A pre-image map for the squared reconstruction error: *Consider the estimation of the reconstruction map for the squared reconstruction error* $||x - \Gamma(\phi(x))||^2$. *A nonlinear map from feature space coordinates* $\boldsymbol{V}_r$ *to the i-th. input dimension* $\mathcal{Z}$ *can be obtained by collecting all i-th dimensions of all patterns in* D_N. *Thus we obtain a vector* $y \in \mathbb{R}^N$ *where the j-th entry of y contains the i-th entry of the sample* $x_j \in D_N$, *i.e.:* $y_j = x_j^i$. *The coefficients* $\boldsymbol{\beta}^i$ *can now be obtained by*

$$\beta^i = \arg\min \sum_{j=1}^{N} \left(y_j - \sum_{r=1}^{N} \beta_r^i \kappa(\boldsymbol{V}_r^\top \phi(x_i), \boldsymbol{V}_r^\top \phi(x_r)) \right)^2 + \lambda||\beta^i||^2.$$

The optimal coefficient vector β^i *is obtained by*

$$\beta^i = (K + \lambda \boldsymbol{I}_N)^\dagger y,$$

where $K = [\kappa(\boldsymbol{V}_r^\top \phi(x_s), \boldsymbol{V}_r^\top \phi(x_t))]_{s,t}$, *and* $1 \leq s, t \leq N$ *denotes the kernel matrix on* D_N *with kernel* κ. *Given some coordinates* $\boldsymbol{V}_r^\top \boldsymbol{\Psi}$, *the i-th entry of the pre-image* z^* *can now be predicted as*

$$z^{*i} = \sum_{s=1}^{N} \beta_s^i y_s \kappa(\boldsymbol{V}_r^\top \phi(x_s), \boldsymbol{V}_r^\top \boldsymbol{\Psi}).$$

Note that the general learning setup allows for using any suitable loss function, incorporating invariants and a-priori knowledge. For example, if the pre-images are (natural) images, a psychophysically motivated loss function could be used which would allow the algorithm to ignore differences that cannot be perceived. Note that the learning based pre-image algorithm requires additional data to learn the reconstruction map similar to MDS and the fixed-point pre-image method. However, this is the only method that does not need any optimization during a testing stage and has an adaptable loss function. Let us now investigate the performance of all pre-image methods on a de-noising experiment.

3.3 Evaluation of pre-image techniques for continuous input spaces

In this section we want to compare the performance of the pre-image techniques for the continuous case in the scenario of KDE. As a task we choose the problem of super-resolution similar to [49]. In contrast to [49] we have a supervised setup: we have given a set of N image pairs $D_N = \{x_i, y_i\}_{i=1}^N$, where the image x_i, y_i show the same content on two different resolutions. Let x_i be the image with the lower resolution. We apply KDE with input and output kernel k, l equal to a Gaussian kernel with the two kernel width parameters σ_i, σ_o. Using this two kernels we apply a kPCA on the input and output images each and formulate afterwards a regression problem using the extracted feature coordinates. Our goal is to use kPCA to build an image model of the low resolution images, and then use a regression model to estimate the coordinates of features corresponding to the higher

Algorithm A3.3 LEARNING THE PRE-IMAGE MAP

Given data $D_N = \{x_i\}_{i=1}^N$, a reconstruction measure $l : \mathcal{Z} \times \mathcal{Z} \to \mathbb{R}_+$, a reconstruction kernel κ and an orthogonal basis $\boldsymbol{V}_r$ for the span of the data D_N in feature space.

Calculate reconstruction models $[\beta^1, \ldots, \beta^3]$ for each dimension of $\mathcal{Z}$ via

$$\beta^j = \arg\min_{\boldsymbol{\beta}^j \in \mathbb{R}^N} \sum_{i=1}^{m} l\left(x_i^j, \Gamma_j(x_i|\boldsymbol{\beta}^j)\right) + \lambda||\boldsymbol{\beta}^j||^2.$$

For a squared loss function, this can be done by kernel ridge regression. See example E3.1.

After training, a pre-image for a point $\boldsymbol{\Psi}$ can be obtained by

1. Extracting coordinates of $\boldsymbol{\Psi}$ by projection $\boldsymbol{V}_r^\top \boldsymbol{\Psi}$.
2. Perform prediction for each dimension i of $\mathcal{Z}$:

$$z^{*i} = \sum_{s=1}^{N} \beta_s^i x_s^i \kappa(\boldsymbol{V}_r^\top \phi(x_s), \boldsymbol{V}_r^\top \boldsymbol{\Psi}).$$

resolution image. After estimation, we need to restore the image from the feature space coordinates, thus we have to solve a pre-image problem. The overview of the experiment is given in figure 3.2.

As images we generate gray-scale images of faces using a virtual human modeler software[3]. where we vary the camera and lighting parameters randomly for each frame. To obtain the small version of a face, we use a resolution 30×30 pixel for rendering where as to yield the higher resolution image we use 60×60 pixels for rendering. We generate 500 image pairs and use 5 randomly chosen images as test set. We use the average of the linear distances of input and output images as kernel widths, which is a reasonable choice since our dataset does not contain any outliers or isolated clusters. Some typical elements of the dataset are shown in figure 3.3.

Since we have used ca. 500 images for training we obtain 500 features by kPCA, however we only keep the first 100 directions since they contain 95% of the variance in

[3]POSER 3D; www.curiouslabs.com

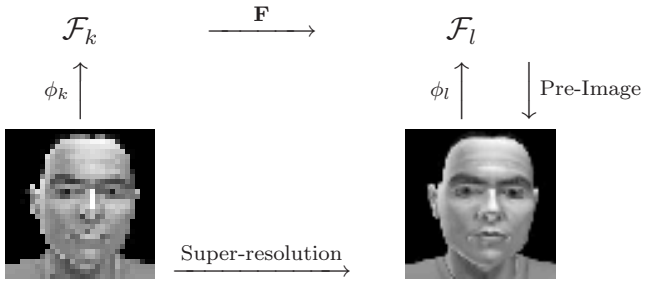


Figure 3.2: Mappings in KDE used for super-resolution. Super-resolution is performed by building statistical models of low and high resolution images using kPCA and estimation coordinates between them. The final image is obtained by solving a pre-image problem.

Figure 3.3: Typical elements from the training set of the super-resolution task.

the feature space. After performing kPCA we use MRS (see chapter 2) to obtain a linear model from input to output features where we have chosen the rank parameter to be 30 which we determined using 5 fold cross validation on the training set. After prediction of the 30 output features we reconstruct the output image which is a 3600 dimensional real vector using the introduced pre-image algorithms: Pre-images by gradient descent, by fixed-point, by multi dimensional scaling and by learning.

For the gradient based as well as for the fixed-point based pre-image approach we have used two restarts. For the MDS method we have used 3 nearest neighbors since increasing the number of neighbors led to a stronger averaging effect and yielded smoothening and smearing effects. For learning we have chosen ridge regression with a Gaussian kernel width σ where the ridge and the kernel width are chosen using 5 fold cross validation on the training set by using the squared reconstruction error as score.

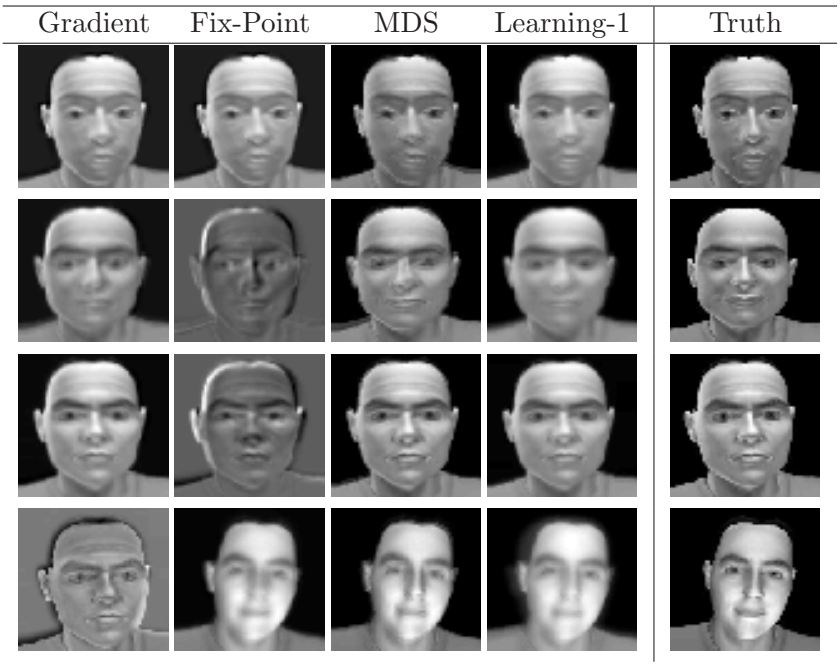


Figure 3.4: Reconstructed images by applying various pre-image techniques.

As one can see in figure 3.4 the gradient descent and fixed-point pre-image methods

can yield results which do not need to be feasible, since they do not take into account the distribution of output patterns. One can see that both techniques return images with partially negative gray scale values indicated by a light gray background. Furthermore, for a high-dimensional pre-image problem the computation of the gradients alone takes very long and thus the gradient descent technique quickly gets intractable. In contrast, the fixed-point method is much quicker. For the performed experiments the gradient descent was 75 times slower than the fixed-point method. However, we can see from the reconstructed images that the algorithms which make explicitly use of additional data are clearly superior in reconstruction. Furthermore, a further pleasant effect is that MDS and the learning based approach are the fastest to evaluate. The gradient based technique was ca. 1400 times slower than the learning based pre-image approach (including model selection) on the current experiment. If we compare MDS and learning based pre-image technique we can see that the learning based approach is a more smoothened version of MDS, since the learning based approach uses a single model which involves all data for prediction and not only the possibly n-nearest neighbors as in MDS. This leads to averaging over more points and thus to the observed smoothing.

We have investigated various approaches for pre-image computation which all are based on one critical assumption: the pre-image space $\mathcal{Z}$ is not-discrete. Currently, there exist no pre-image method for the case that the pre-image is of discrete nature. To this end, let us investigate in the following sections the pre-image problem for complex objects as strings and graphs being discrete.

3.4 Pre-Images for Complex Objects

The pre-image problem is particularly challenging when the target set $\mathcal{Z}$ has discrete nature. For example consider the application of KDE in the setting of machine translation: one has to map from an input sequence of words to an output sequence of words. The pre-image is a sequence of words. Alternatively consider the problem of estimating an endomorphism on the set of graphs given some example input output graph pairs. Here, the pre-image is a graph. In this case, the problem P3.3 becomes a combinatorial problem and thus gets very difficult to solve. There exists techniques for the case that the pre-image is a sequence which require a very special feature space structure. In contrast, our approach could in principle be applicable for *any* kernel suitable on $\mathcal{Z}$. However in our experiments we will use one particular kernel introduced by [46].

Our optimization technique is mainly motivated on randomization principles found in [66] and on a novel framework for stochastic optimization called *the cross-entropy method* which was introduced by [74].

3.4.1 From combinatorial optimization to estimation

In the following we discuss the cross-entropy (CE) method for combinatorial optimization where we mainly follow [74]. Consider the following minimization problem: Let $\mathcal{Z}$ be a finite set of states, and let $\mathcal{L}$ be a real valued cost function on $\mathcal{Z}$. We want to find *a subset of states* with elements z^* minimizing $\mathcal{L}$. Let γ^* denote the attained minimum of $\mathcal{L}$, i.e.:

$$\mathcal{L}(z^*) = \gamma^* = \min_{z \in \mathcal{Z}} \mathcal{L}(z). \tag{17}$$

The key idea of the CE method is to associate with optimization problem (17) a new *estimation* problem. To this end, we first define a collection of indicator functions $\{1_{\{\mathcal{L}(z) \leq \gamma\}}\}$ on $\mathcal{Z}$ for various levels $\gamma \in \mathbb{R}$. Next, let $\{f(\cdot; \boldsymbol{v})\}, \boldsymbol{v} \in \mathcal{V}$ be a family of discrete probability

densities on $\mathcal{Z}$, parameterized by a real-valued parameter vector $\mathbf{v} \in \mathcal{V}$ where $\mathcal{V}$ denotes the set of feasible parameter vectors. For a given fixed $\boldsymbol{u} \in \mathcal{V}$ we are now able to associate with problem (17) the problem of estimating the probability $l(\gamma)$ of minimizing the cost function $\mathcal{L}$, i.e.:

$$l(\gamma) = \mathbb{P}_{\boldsymbol{u}}(\mathcal{L}(\boldsymbol{Z}) \leq \gamma) = \mathbb{E}_{\boldsymbol{u}} 1_{\{\mathcal{L}(\boldsymbol{Z}) \leq \gamma\}} = \int 1_{\{\mathcal{L}(Z) \leq \gamma\}} f(Z; \boldsymbol{u}) dZ, \tag{18}$$

where $\mathbb{P}_{\boldsymbol{u}}$ is the probability measure under which the random state $\boldsymbol{Z}$ has pdf $f(\cdot; \boldsymbol{u})$, and $\mathbb{E}_{\boldsymbol{u}}$ denotes the expectation regarding the pdf $f(\cdot; \boldsymbol{u})$. A possible way to estimate $l(\gamma)$ would be to use Monte Carlo simulation techniques (see e.g. [56]). That is, drawing a random sample $\{\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_N\}$ from the distribution $f(\cdot; \boldsymbol{u})$ and use

$$\frac{1}{N} \sum_{i=1}^{N} 1_{\{\mathcal{L}(\boldsymbol{Z}_i) \leq \gamma\}}$$

as an unbiased estimator of $l(\gamma)$. Note that γ is small for the vicinity of minima of (17). Therefore we aim at estimating the probability of a *rare* event. In this case large γ's and small values of the probability density function will dominate and unacceptably many points will have to be generated by Monte Carlo. In other words, the estimate will have large variance and in fact Crude Monte Carlo simulations are likely to be high variance estimators (see [74] for a discussion). An efficient way to reduce the variance of our estimate $l(\gamma)$ is to propose an *importance sampling density* $f(\cdot; \boldsymbol{v})$ and to express l in terms of $f(\cdot; \boldsymbol{v})$ as

$$l(\gamma) = \mathbb{E}_{\boldsymbol{v}} 1_{\{\mathcal{L}(\boldsymbol{Z}) \leq \gamma\}} \frac{f(\boldsymbol{Z}; \boldsymbol{u})}{f(\boldsymbol{Z}; \boldsymbol{v})}. \tag{19}$$

An unbiased estimator of $l(\gamma)$ can now be determined using

$$l(\gamma) = \frac{1}{N} \sum_{i=1}^{N} 1_{\{\mathcal{L}(\boldsymbol{Z}_i) \leq \gamma\}} W(\boldsymbol{Z}_i; \boldsymbol{u}, \boldsymbol{v}),$$

where $W(\boldsymbol{Z}_i; u, v) = \frac{f(\boldsymbol{Z}; \boldsymbol{u})}{f(\boldsymbol{Z}; \boldsymbol{v})}$ is called the likelihood ratio, and $\{\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_N\}$ are now random samples from $f(\cdot; \boldsymbol{v})$. This *change of measure* allows us now to choose the parameter vector $\boldsymbol{v}$ such that the variance of our estimate is small, i.e.:

$$\boldsymbol{v}^* = \arg\min_{\boldsymbol{v} \in \mathcal{V}} \mathbb{E}_{\boldsymbol{v}} \left[1_{\{\mathcal{L}(\boldsymbol{Z}) \leq \gamma\}} W(\boldsymbol{Z}; \boldsymbol{u}, \boldsymbol{v}) - \mathbb{E}_{\boldsymbol{v}} 1_{\{\mathcal{L}(\boldsymbol{Z}) \leq \gamma\}} W(\boldsymbol{Z}; \boldsymbol{u}, \boldsymbol{v}) \right]^2. \tag{20}$$

Unfortunately, the optimal reference parameter $\boldsymbol{v}^*$ is seldom available analytically, since the expectation $\mathbb{E}_{\boldsymbol{v}} 1_{\{\mathcal{L}(\boldsymbol{Z}) \leq \gamma\}} W^2(\boldsymbol{Z}; \boldsymbol{u}, \boldsymbol{v})$ cannot be evaluated exactly in the most cases. An alternative to the variance minimization approach for estimating the optimal reference parameter vector $\boldsymbol{v}$ is based on the Kullback-Leibler divergence which defines a 'distance' between two pdfs f and g and is given by

$$\mathcal{D}(f, g) = \int g(x) \ln \frac{g(x)}{f(x)} d\boldsymbol{x} = \int g(x) \ln g(x) d\boldsymbol{x} - \int g(x) \ln f(x) d\boldsymbol{x}.$$

Let us first consider the theoretically optimal importance sampling density g^* which follows from minimizing the non-parametric pendant to (20). It is the variational problem

$$g^* = \arg\min_{g} \mathbb{E}_g \left[1_{\{\mathcal{L}(\boldsymbol{Z}) \leq \gamma\}} \frac{f(\boldsymbol{Z}; \boldsymbol{u})}{g(\boldsymbol{Z})} - \mathbb{E}_g 1_{\{\mathcal{L}(\boldsymbol{Z}) \leq \gamma\}} \frac{f(\boldsymbol{Z}; \boldsymbol{u})}{g(\boldsymbol{Z})} \right]^2,$$

which can be shown to be equivalent to

$$g^* = \arg\min_g \mathbb{E}_g \left[1_{\{\mathcal{L}(\boldsymbol{Z}) \leq \gamma\}} \frac{f(\boldsymbol{Z}; u)}{g(\boldsymbol{Z})} \right] - l(\gamma).$$

It is easy to see that the optimal density is given by

$$g^* = \frac{f(\boldsymbol{Z}; \boldsymbol{u}) 1_{\{\mathcal{L}(\boldsymbol{Z}) \leq \gamma\}}}{l(\gamma)}. \tag{21}$$

Obviously this density is useless, since it requires the knowledge of $l(\gamma)$. However, given our parametric family of densities $\{f(\cdot; \boldsymbol{v})\}$ we can use the KL divergence to setup up the parametric optimization problem

$$\boldsymbol{v}^* = \arg\min_{\boldsymbol{v} \in \mathcal{V}} \mathcal{D}(g^*, f(\cdot; \boldsymbol{v})).$$

Now using the theoretically best possible proposal density g^* given in (21), we obtain the optimization problem

$$\begin{aligned}
\arg\min_{\boldsymbol{v} \in \mathcal{V}} \mathcal{D}(g^*, f(\cdot; \boldsymbol{v})) &= \arg\min_{\boldsymbol{v} \in \mathcal{V}} \int g^*(Z) \ln g^*(Z) dZ - \int g^*(Z) \ln f(Z; \boldsymbol{v}) dZ \\
&= \arg\max_{\boldsymbol{v} \in \mathcal{V}} \int g^*(Z) \ln f(Z; \boldsymbol{v}) dZ \\
&= \arg\max_{\boldsymbol{v} \in \mathcal{V}} \int f(Z; \boldsymbol{u}) \frac{1_{\{\mathcal{L}(Z) \leq \gamma\}}}{l(\gamma)} \ln f(Z; \boldsymbol{v}) dZ \\
&= \arg\max_{\boldsymbol{v} \in \mathcal{V}} \mathbb{E}_{\boldsymbol{u}} 1_{\{\mathcal{L}(Z) \leq \gamma\}} \ln f(Z; \boldsymbol{v}) d\boldsymbol{Z}. \qquad (22)
\end{aligned}$$

Thus, the only relevant part of the KL divergence is the *cross-entropy*. Note that (22) does not contain any unknowns but just requires the evaluation of the integral. To this end we can replace it by a stochastic version and consider solving

$$\boldsymbol{v}^* = \arg\max_{\boldsymbol{v} \in \mathcal{V}} \frac{1}{N} \sum_{i=1}^{N} 1_{\{\mathcal{L}(\boldsymbol{Z}_i) \leq \gamma\}} \ln f(\boldsymbol{Z}_i; \boldsymbol{v}). \tag{23}$$

What do we gain by doing so? The advantage of problem formulation (23) compared to (22) is that for a wide range of density models we can calculate the optimal parameter corresponding to the minimum KL divergence *analytically*!

Namely, the optimal parameter vector $\boldsymbol{v}^*$ is given now by the solution of the linear system

$$\frac{\partial}{\partial \boldsymbol{v}_i} \frac{1}{N} \sum_{i=1}^{N} 1_{\{\mathcal{L}(\boldsymbol{Z}_i) \leq \gamma\}} \ln f(\boldsymbol{Z}_i; \boldsymbol{v}) = 0, \; 1 \leq i \leq |v|.$$

To illustrate the advantage of the CE method consider the example of estimating parameter vectors of a Bernoulli pdf for a combinatorial problem.

Example E3.2 Optimal importance sampling parameter for Bernoulli random variables: *Given that $\boldsymbol{Z} \in \{0,1\}^d$ denotes a d dimensional bit vector and $f(\boldsymbol{Z}; \boldsymbol{v}) = \prod_{i=1}^{d} v_i^{Z_i} (1 - v_i)^{1 - Z_i}$ denotes the pdf of the random vector $\boldsymbol{Z}$. We want to solve the stochastic problem (23) given a set $\{\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_N\}$. Taking derivatives and considering the extreme*

point results in

$$\begin{aligned}
0 &= \frac{\partial}{\partial \boldsymbol{v}_k} \frac{1}{N} \sum_{j=1}^{N} 1_{\{\mathcal{L}(\boldsymbol{Z}_j) \leq \gamma\}} \ln f(\boldsymbol{Z}_j; \boldsymbol{v}), \\
0 &= \sum_{j=1}^{N} 1_{\{\mathcal{L}(\boldsymbol{Z}_j) \leq \gamma\}} \frac{\partial}{\partial \boldsymbol{v}_k} \left(\sum_{i=1}^{d} \ln \left(v_i^{Z_{ji}} (1 - v_i)^{1 - Z_{ji}} \right) \right), \\
0 &= \sum_{j=1}^{N} 1_{\{\mathcal{L}(\boldsymbol{Z}_j) \leq \gamma\}} \frac{\partial}{\partial \boldsymbol{v}_k} \left(\ln \left(v_k^{Z_{jk}} \right) + \ln \left((1 - v_k)^{1 - Z_{jk}} \right) \right), \\
0 &= \sum_{j=1}^{N} 1_{\{\mathcal{L}(\boldsymbol{Z}_j) \leq \gamma\}} \left(\frac{Z_{jk}}{v_k} - \frac{1 - Z_{jk}}{1 - v_k} \right),
\end{aligned}$$

and thus the optimal parameter v_k *for* $1 \leq k \leq d$ *is given by*

$$v_k = \frac{\sum_{j=1}^{N} 1_{\{\mathcal{L}(\boldsymbol{Z}_j) \leq \gamma\}} Z_{jk}}{\sum_{j=1}^{N} 1_{\{\mathcal{L}(\boldsymbol{Z}_j) \leq \gamma\}}}.$$

We can now formulate a simple iterative algorithm where we tune the level parameter γ, estimate the probabilities and determine the next sampling parameters based on KL divergence. The sampling parameters are updated whenever the level parameter is lower than the level parameter from a previous iteration. The algorithm is described in A3.4.

Our ultimate goal is to apply the CE method for calculating pre-images of graphs and sequences. Theoretically we can apply the CE method to pre-image problems with any graph and string kernels, however in general we expect pre-images with special properties, thus we have a prior belief about the class of relevant pre-images. Assume thus that we have given such a prior on objects, how can we extend the CE method to use it?

3.4.2 Adding a Prior to the CE Method

Once the optimal importance distribution parameters $\boldsymbol{v}^k$ are estimated we can sample from the distribution $f(\cdot; \boldsymbol{v}^k)$ and continue the optimization process. However, [74] remarks that if the update rule is modified to

$$\boldsymbol{v}_k = \alpha \boldsymbol{v}^k + (1 - \alpha) \boldsymbol{v}^{k-1},$$

better performance can be observed. [74] do not motivate this update rule but argue that this *momentum* term leads to a smoothing of the estimate $\boldsymbol{v}$. Let us use the opportunity to go further.

By considering that $\boldsymbol{v}^k$ is an estimate $F(Z)$ from the data we can rewrite the smoothing equation as

$$\boldsymbol{v}^k = \alpha F(Z) + (1 - \alpha) \boldsymbol{v}^{k-1},$$

and it is easy to check that the resulting $\boldsymbol{v}_k$ is the solution to the optimization problem

$$\boldsymbol{v}^k = \arg\min_{\boldsymbol{v}} \ ||\alpha F(Z) - \frac{1}{2}\boldsymbol{v}||^2 + ||(1 - \alpha)\boldsymbol{v}^{k-1} - \frac{1}{2}\boldsymbol{v}||^2. \tag{24}$$

Note that one can interpret this as

$$\boldsymbol{v} = \arg\max_{\boldsymbol{v}} P_1(F(Z)|\boldsymbol{v}) P_2(\boldsymbol{v}|\boldsymbol{v}^{k-1}), \tag{25}$$

Algorithm A3.4 CE ALGORITHM FOR COMBINATORIAL OPTIMIZATION (SEE [74])

Given the number N of data points to sample, a family of distributions $\{f(\cdot;\boldsymbol{v})\}$ and the quantile parameter $0 < \varrho < 1$.

Initialization

1. $k = 1$, choose some initial parameter sampling $\boldsymbol{v}^{(0)}$.

While $||\boldsymbol{v}^k - \boldsymbol{v}^{k-1}|| > \epsilon$ and an update happened in the last t iterations

2. Generate a random sample $\{\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_N\}$ according the sampling distribution $f(\cdot;\boldsymbol{v}^{k-1})$.
3. Determine the minimization level parameter γ^k. In [74], it was proposed to sort the $\boldsymbol{Z}_i$ descending according to their loss $\mathcal{L}(\boldsymbol{Z}_i)$, and set γ^k to $\mathcal{L}(\boldsymbol{Z}_{\lceil(1-\varrho)N\rceil})$.
4. If $(\gamma^k < \gamma^{k-1})$
5. Determine the optimal importance sampling distribution parameters $\boldsymbol{v}^k$ by solving

$$\boldsymbol{v}^k = \arg\max_{\boldsymbol{v}\in\mathcal{V}} \frac{1}{N}\sum_{i=1}^{N} 1_{\{\mathcal{L}(\boldsymbol{Z}_i)\leq\gamma\}} \ln f(\boldsymbol{Z}_i;\boldsymbol{v}),$$

6. k=k+1

with appropriate distribution functions P_1, P_2. We can therefore think of the new update rule as maximum a-posteriori estimate of the parameters under the prior $P_2(\boldsymbol{v}|\boldsymbol{v}^{k-1})$. Therefore, it is theoretically straightforward to replace this "continuity" prior by one which takes into account additional data, thus a data dependent prior. Unfortunately, to be computationally feasible one has to select a *conjugated prior* which is mostly not appropriate for the distribution at hand. In such cases it is still possible to use data to select a reasonable starting value $\boldsymbol{v}^0$ and initialize the solution path for the smoothing prior.

Since we have introduced our main computational tool, we are now ready to introduce our kernel of main interest: the marginalized kernel for sequences and graphs.

3.4.3 The marginalized kernel for sequences and graphs

In this section we discuss the use of the marginalized kernel which is suited to handle discrete data structures like directed labeled graphs and sequences being a special case of labeled graphs. In the following we consider the general case first: node-labeled, undirected graphs $z = (v, E)$, where the nodes are identified with the set $V = \{1, \ldots, |z|\}$, $|z|$ denoting the order of the graph, the function $v : V \to \Sigma$ supplies the labels of the nodes which are taken from some finite set Σ, and $E \in \{0,1\}^{|z|\times|z|}$ is the adjacency matrix, i.e. $e(i,j) = 1$ if there is an edge between nodes i and j. z denotes the set of all possible graphs.

Even though the matrix E at first sight seems to suggests the use of classical kernels that take real vectors as inputs, there are two severe problems to this approach:

(ii) Different matrices E can represent the same graph, since the nodes can be re-ordered.

(ii) We want to be able to work with graphs of different sizes.

It was therefore proposed in [46] to compare two graphs by measuring the similarity of the probability distributions of label sequences generated by random walks on the two

graphs. By using the dot product of the two probability distributions as kernel, the induced feature space $\mathcal{F}$ is infinite dimensional, with one dimension for every possible label sequences. Nevertheless, the authors developed an efficient way to calculate the dot product explicitly.

Note that only a subset of $\mathcal{F}$ correspond to realizable graphs z. First, the feature space image $\phi(z)$ of any graph z must be a valid probability distribution, i.e. its components must be non-negative and sum to one. While this is always satisfied by convex combinations of graph images, arbitrary linear combinations may lead to negative components. Further conditions for the existence of a pre-image results from constraints on the probabilities (if some label sequence has positive probability, then so have its subsequences) and the finiteness of the graphs (only a countable set of possible probabilities).

We briefly review the marginalized graph kernel introduced in [46], basically following the notation used there while omitting edge labels. Let $\Omega(z)$ denote the set of all possible node paths $\boldsymbol{h}$ on a graph $z = (V, E)$, i.e. $\boldsymbol{h} \in V^n$ statistics $E(\boldsymbol{h}_i, \boldsymbol{h}_{i+1}) = 1$ for every $i < n$; $|\boldsymbol{h}| := n$ is the path length. We define a probability distribution $p(\boldsymbol{h}|z)$ over $\Omega(z)$ by considering random walks with start probability $p_s(i)$ at node i, a constant termination probability $p_q(i) = \lambda$ and a transition probability $p_t(i, j)$ which is positive only for edges, i.e. if $E(i, j) = 1$.

Realizing the fact, that a path is a sequence of labels corresponding to connected vertices, we can define the following kernel for element-wise comparison of paths with same length:

$$k_h(\boldsymbol{h}, \boldsymbol{h}') \quad = \quad \begin{cases} \delta(v_{\boldsymbol{h}_1}, v_{\boldsymbol{h}'_1}) \prod_{k=2}^{|\boldsymbol{h}|} \delta(v_{\boldsymbol{h}_k}, v_{\boldsymbol{h}'_k}) \delta(e_{\boldsymbol{h}_{k-1}, \boldsymbol{h}_k}, e'_{\boldsymbol{h}_{k-1}, \boldsymbol{h}_k}) & |\boldsymbol{h}| = |\boldsymbol{h}'| \\ 0 & |\boldsymbol{h}| \neq |\boldsymbol{h}'| \end{cases}$$

where $\delta(a, b)$ is 1 for $a = b$ and otherwise 0. Note that in general, it is possible to replace the hard δ function with *softer* kernel functions $k_v(v, v'), k_e(e_{ij}, e'_{ij}) \geq 0$.[4] However we will restrict the discussion to the δ function only. Now any kernel k_h on node paths induces a kernel on graphs via

$$k(z, z') = \mathbb{E}_{\boldsymbol{h}, \boldsymbol{h}'}[k_h(\boldsymbol{h}, \boldsymbol{h}')] \quad = \quad \sum_{\boldsymbol{h} \in \Omega(z)} \sum_{\boldsymbol{h}' \in \Omega(z')} k_h(\boldsymbol{h}, \boldsymbol{h}') p(\boldsymbol{h}|z) p(\boldsymbol{h}'|z'), \tag{26}$$

which can be interpreted as the average similarity over all random walks $\boldsymbol{h}, \boldsymbol{h}'$ occurring in the graphs z, z'.

To calculate the conditional distributions $p(\boldsymbol{h}|z), p(\boldsymbol{h}'|z')$ for the random walk $\boldsymbol{h}, \boldsymbol{h}'$, we need to define a start, a transition and a termination probability distribution over the vertices in Σ_v of $\boldsymbol{h}$. The start of the random walk $\boldsymbol{h}$ is determined by the probability distribution $p_s(\boldsymbol{h}_1)$. Subsequently, at the i-th step, the walk might end with probability $p_e(\boldsymbol{h}_i)$ or perform a transition to the next vertex $\boldsymbol{h}_i$ with probability $p_t(\boldsymbol{h}_i|\boldsymbol{h}_{i-1})$. Therefore the conditional probability for a path $\boldsymbol{h}$ is described as

$$p(\boldsymbol{h}|z) \quad = \quad p_s(\boldsymbol{h}_1) \prod_{i=2}^{|\boldsymbol{h}|} p(\boldsymbol{h}_i|\boldsymbol{h}_{i-1}) p_e(\boldsymbol{h}_i). \tag{27}$$

Note, the similarity to the path kernel structure. Since the graph kernel is the expectation of $k_h(\boldsymbol{h}, \boldsymbol{h}')$, it involves the evaluation of an infinite sum, making it impossible to evaluate

[4] For example, if $v \in \Sigma_v = \mathbb{R}$ the Gaussian kernel $k_v(v_i, v_j) = e^{-\|v_i - v_j\|^2}$ can be used.

it directly. However by using (27) and re-ordering we can write (26) as

$$k(z,z') = \lim_{L\to\infty}\sum_{l=1}^{L}\sum_{\boldsymbol{h}\in\Omega(z)}\sum_{\boldsymbol{h}'\in\Omega(z')} s(\boldsymbol{h}_1,\boldsymbol{h}'_1)\times\prod_{k=2}^{l} t(\boldsymbol{h}_k,\boldsymbol{h}_{k-1},\boldsymbol{h}'_k,\boldsymbol{h}'_{k-1})\times q(\boldsymbol{h}_k,\boldsymbol{h}'_k) \tag{28}$$

with

$$\begin{aligned} s(\boldsymbol{h}_1,\boldsymbol{h}'_1) &= \delta(v_{\boldsymbol{h}_1},v_{\boldsymbol{h}'_1})p_s(\boldsymbol{h}_1)p_s(\boldsymbol{h}'_1),\\ t(\boldsymbol{h}_k,\boldsymbol{h}_{k-1},\boldsymbol{h}'_k,\boldsymbol{h}'_{k-1}) &= \delta(v_{\boldsymbol{h}_k},v_{\boldsymbol{h}'_k})\delta(e_{\boldsymbol{h}_{k-1},\boldsymbol{h}_k},e'_{\boldsymbol{h}_{k-1},\boldsymbol{h}_k})p(\boldsymbol{h}_k|\boldsymbol{h}_{k-1})p(\boldsymbol{h}'_k|\boldsymbol{h}'_{k-1}),\\ q(\boldsymbol{h}_i,\boldsymbol{h}'_j) &= p_e(\boldsymbol{h}_i)p_e(\boldsymbol{h}'_j). \end{aligned}$$

The matrices s, q have the shape $\boldsymbol{R}^{|z|\times|z'|}$, where as t is of size $\boldsymbol{R}^{|z|\times|z|\times|z'|\times|z'|}$. Now, if we sort the sums in (28) over the index in the path we unveil the following structure

$$k(z,z') = \sum_{\boldsymbol{h}_1}\sum_{\boldsymbol{h}'_1} s(\boldsymbol{h}_1,\boldsymbol{h}'_1)\times R(\boldsymbol{h}_1,\boldsymbol{h}'_1)$$

with

$$R(\boldsymbol{h}_1,\boldsymbol{h}'_1) = \lim_{L\to\infty}\sum_{l=1}^{L}\sum_{\boldsymbol{h}_2}\sum_{\boldsymbol{h}'_2} t(\boldsymbol{h}_2,\boldsymbol{h}_1,\boldsymbol{h}'_2,\boldsymbol{h}'_1)\times\left(\ldots\left(\sum_{\boldsymbol{h}_l}\sum_{\boldsymbol{h}'_l} t(\boldsymbol{h}_l,\boldsymbol{h}_{l-1},\boldsymbol{h}'_l,\boldsymbol{h}'_{l-1})\times q(\boldsymbol{h}_l,\boldsymbol{h}'_l)\right)\right).$$

The convergence conditions were investigated in [46], and are shown to be quite weak. Particularly, it is required that the used vertex kernel $k_v(v_i,v_j)$ and the edge label kernel $k_e(e_{ij},e'_{ij})$ are positive and less than or equal to 1. This is trivially the case for the used δ function. It was further shown that on convergence the following equilibrium condition is fulfilled:

$$R(\boldsymbol{h}_1,\boldsymbol{h}'_1) = q(\boldsymbol{h}_1,\boldsymbol{h}'_1) + \sum_{i=1}^{|z|}\sum_{j=1}^{|z'|} t(i,\boldsymbol{h}_1,j,\boldsymbol{h}'_1)R(\boldsymbol{h}_1,\boldsymbol{h}'_1). \tag{29}$$

This is a linear equation system which can be solved. By reshaping the matrices s, q, r to vectors $\boldsymbol{s},\boldsymbol{q},\boldsymbol{r}\in\mathbb{R}^{|z||z'|}$, and rewriting the tensor t as a matrix $T\in\mathbb{R}^{|z||z'|\times|z||z'|}$ the graph kernel can be expressed as:

$$k(z,z') = \boldsymbol{s}^\top(\mathbb{1}-T)^{-1}\boldsymbol{q}. \tag{30}$$

Since we have introduced our main similarity measure of interest, let us return to the question of calculating pre-images. Solving the pre-image problem for sequences and graphs requires obviously to a) determine the vertex label set $\Sigma_v^*\in z^*$ and b) to determine the adjacency matrix $E^*\in z^*$. We will begin our discussion in the next section with the case of sequences, since this is a simpler problem due the serial structure of sequences. Then, we proceed with the reconstruction of graph pre-images, where we will use ideas from sequence pre-images reconstruction.

3.4.4 Pre-images for sequences

Some applications require to predict fixed-length sequences only such as part-of-speech tagging or protein secondary structure prediction. In these tasks the target sequence

length $m^* := |z^*|$ is known (it is the same length as the input). However, for most applications the predicted sequence length m^* is unknown. Therefore let us consider the case where the output space $\mathcal{Z}$ consists of sequences with finite but unknown length and where the marginalized graph kernel is for sequences (see figure 3.5 for illustration). Furthermore, let us introduce the symbol $\epsilon \in \Sigma_v$ which denotes the beginning of a word, and let $|\Sigma_v| =: Z$ be the number of all possible symbols. The labels v appearing in the sequence are elements of some basic alphabet Σ_v, and we declare ϵ to be the starting symbol of every sequence.

The adjacency matrix defines the connectivity of a sequence and trivially for sequences only the entries $E(i, i+1)$ with $1 \leq i < m^*$ are non-zero. It is easy to show that adjacency matrices defined in such a manner are nil-potent, thus there exist a $k > 1$ such that

$$E^k = 0.$$

Thus to reconstruct the sequence pre-image we need

- to determine the length of the sequence,
- to know the subset of the alphabet $\Sigma_v^* \subset \Sigma_v$ which is used to construct the sequence $z^* \in \mathcal{Z}$, and
- to determine the order of the symbols, thus to estimate the permutation of the symbols in Σ_v^*.

Let us first discuss the issue of determining the sequence length.

The length of z^*: It may come as a surprise that the marginalized kernel values do not contain information about the sequence size. To see why this is the case, consider an easy example. Fix an arbitrary sequence z and its kernel values with other sequences. Now consider the sequence $2z$ which is defined by duplicating z, i.e. it consists of two unconnected copies of z. Thus, the start probabilities (for each copy of each node) are divided by two, while the transition and termination probabilities remain unchanged. Thus, for each of the two copies, the feature space representation (the histogram of label paths) is also divided by two. Adding the histograms of the two copies (which corresponds to use the combined sequence) recovers the original feature space representation. Therefore, all kernel values have to be the same. Since the kernel does not help in determining the size, we have to make use of heuristics. We consider three simple ideas to fix $m^* := |z^*|$, the size of z^*:

i) *Linear combination of example sequence sizes:*
If the point $\mathbf{\Psi}$ is given as an expansion, one intuitive idea is to determine m^* as a linear combination of the size of the sequences specifying $\mathbf{\Psi}$. It is natural to give each sequence the same amount (α_i) of influence on the size of z as it has on z. Thus we have the weighted average

$$m^* = \frac{\sum_{i=1}^{N} \alpha_i m_i}{\sum_{i=1}^{N} \alpha_i}, \tag{31}$$

where m_i is the size of sequence z_i. If $\mathbf{\Psi}$ is given by its coordinates only, one could use the coordinate distances $||\boldsymbol{V}_r^\top \mathbf{\Psi} - \boldsymbol{V}_r^\top \phi(x_i)||$ of $\mathbf{\Psi}$ and some x_i for interpolation.

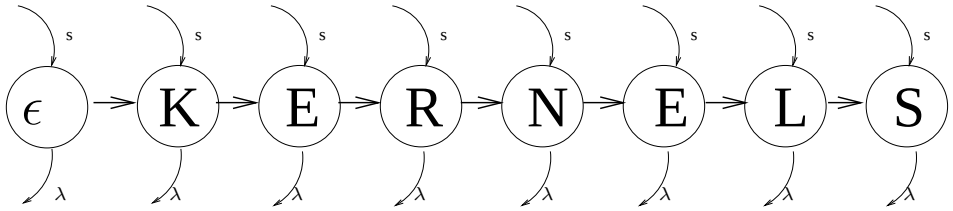


Figure 3.5: Illustration of all paths which are used for comparison when using the marginalized kernel on the sequence "K-E-R-N-E-L-S". A path can start on any node with probability s and quit on any node with probability λ.

ii) *Exhaustive search in a range:*
For exhaustive search we need to restrict ourselves to a finite range of plausible sizes, for example between minimum and maximum size of example sequences. Then a tentative sequence is reconstructed for each size in the range, and the one best approximating ψ is chosen afterwards.

iii) *Regression:*
The most sophisticated approach is to *learn* the sequence size. Since in KDE we have example sequences given in D_N, we can state the order estimation as a regression problem: Using the kernel map, we explicitly construct features $x_i,\ 1 \leq i \leq N$ by $x_i = (k(z_i, z_1), \ldots, k(z_i, z_N)) \in \mathbb{R}^N$. Letting $y_i = |z_i|$, we can apply any regression method to estimate their dependence on the x.

Learning the sequence length has the advantage that it does not use any knowledge of the used kernel itself, but has the disadvantage to be the computationally quit expensive.

Once the size $|z^*|$ is estimated, we are able to determine the vertex labels.

Determining the alphabet Σ_v^*: Determining the alphabet Σ_v^* requires to decide if a label $v_i \in \Sigma_v$ is a member of Σ_v^*, and if so, how many times it appears in the sequence. To answer these two questions, we will use special properties of the marginalized kernel. We introduce the *trivial* sequence, z_{v_i}, which consists just of one vertex $v_i \in \Sigma$ and zero edges. A random walk on the trivial sequence z_{v_i} creates a single path of length 1 consisting of the single vertex label v_i itself. Reconsidering the terms appearing in the marginalized kernel, one sees that the only nonzero terms are

$$\begin{aligned} s(\boldsymbol{h}_1 = v_i, \boldsymbol{h}'_1 = v_i) &= p_s(\boldsymbol{h}'_1 = v_i), \\ q(\boldsymbol{h}_1 = v_i, \boldsymbol{h}'_1 = v_i) &= p_e(\boldsymbol{h}_1 = v_i) p_e(\boldsymbol{h}'_1 = v_i), \end{aligned}$$

while $T \in \mathbb{R}^{|z| \times |z|}$ becomes a zero matrix. Assuming a constant termination probability $p_e(v_k) = \lambda$ and uniform start probabilities $p_s(\boldsymbol{h}_1 = v_k) = 1/m_i$, the evaluation of the marginalized kernel yields

$$\begin{aligned} k(z_{v_k}, z_i) &= m_{ik} \cdot p_s(\boldsymbol{h}_i = v_k) \cdot p_e^2(v_k) = m_{ik} \cdot \lambda^2 / m_i, \\ k(z_{v_k}, z^*) &= m_k^* \cdot p_s(\boldsymbol{h}^* = v_k) \cdot p_e^2(v_k) = m_k^* \cdot \lambda^2 / m^*, \end{aligned}$$

where m_{ik} and m_k^* denote the numbers of occurrence of the label v_k in the graph z_i and z^*, respectively.

We are now able to find the vertex set by solving for m_k^*:

$$m_k^* = k(z_{v_k}, z^*) \cdot m^* / \lambda^2 = \frac{m^*}{\lambda^2} \sum_{i=1}^{N} \alpha_i \lambda^2 \frac{m_{ik}}{m_i} = m^* \sum_{i=1}^{N} \alpha_i \frac{m_{ik}}{m_i}. \tag{32}$$

The last equality shows that the fractions m_k^*/m^* of labels in z^* are just the linear combinations of the fractions in the input sequences. This can be combined with any preset order m^* of z^*.

Determining the vertex set Σ_v^* and the length m^*, we have to answer the last question which is the sorting of the symbols – the sequence ordering.

Determining the symbol ordering Let us construct a lexically ordered sequence z from the elements of Σ_v^*. Then our goal is to determine the right sequence ordering of z such that ideally the re-ordered sequence is the searched pre-image z^*. Thus we are looking for a permutation π^* which permutes the sequence z into z^*. We therefore aim to solve the optimization problem

$$\pi^* = \arg\min_{\pi \in S_m} ||\Psi - \phi(\pi(\Sigma^*))||^2,$$

with π is a permutation operator on m^* elements and S_m denotes the permutation group of order m.

To apply the CE method to perform optimization over permutation groups, we need to specify a family of probability functions $\{f(\cdot; v)\}$ such that sampling from these probability functions generates valid elements of S_m. To this end, let us define first the direct representation of a permutation operator π:

Definition D3.2 Direct Representation of a Permutation.: *Let $\pi \in S_m$ be a permutation operator on m elements, then the direct representation of the permutation operator is given by the image of π applied to the sorted sequence $1, \ldots, m$. Thus the direct representation is given by a sequence of indices indicating their position in the final permuted sequence.*

For example the permutation $\pi = 1, \ldots, m$ specifies the identity element in the group S_m, where $\pi = m, m-1, \ldots, 1$ specifies a permutation which reverses the ordering of a sequence. Besides being intuitive, the direct representation of a permutation is not very convenient for optimization purposes since every element $\pi(x)_i$ in the sequence $\pi(x) = \pi(x)_1, \ldots, \pi(x)_m$ depends on a different element $\pi(x)_j, j \neq i$. For example, any index can appear only once. Thus, we can not simply specify a distribution function where each element of π is sampled from, since these are not independent. To this end we propose to use a different representation of permutations, where we basically follow [52]. Remember that the permutation group S_m has $m!$ elements. If we enumerate all operators $\pi^i \in S_m, 1 \leq i \leq m!$, we can associate with each permutation $\pi \in S_m$ a vector $\boldsymbol{p} \in P \subset \mathbb{N}_0^m$ if the i-th entry p_i of the vector $\boldsymbol{p}$ is constraint to be positive or equal zero and less then i. Note that the parameters can encode all different $m!$ permutations since the number of elements in P is

$$|P| = \max_{\{p_1, \ldots, p_m\}} \prod_{i=1}^{m} (p_i + 1) = \prod_{i=1}^{m} \max_{p_i} (p_i + 1) = m(m-1)\ldots 1 = m!$$

We can embed P into the real space $\mathbb{R}^m$ and always project elements of $\mathbb{R}^m$ back to P by rounding coordinates to the next integer. Now, if we can find a mapping from a direct representation of a permutation to P and back, we could specify distribution functions for each element of the coordinate vector in P independently which is a much simpler task then defining a distribution function for elements of the direct representation. One possible encoding scheme introduced in [92] is

$$p_i = \sum_{j=1}^{i} 1_{\{v_j > v_i\}},$$

which simply counts the number of elements which are predecessors and are lexically larger.

Example E3.3 From the Permutation Group to Vector Space: *Consider the sequence* $x = \{K, e, r, n, e, l, s\}$*, and its lexically sorted form* $\hat{x} = \{K, e, e, l, n, r, s\}$*. We can encode the required permutation* $\pi = (1, 2, 3, 4, 5, 6, 7) \rightarrow (1, 2, 5, 6, 3, 4, 7)$ *which maps* $\hat{x}$ *to* x *in the space* P *as* $\boldsymbol{p} = (0, 0, 2, 2, 1, 0, 0)$.
To retrieve the permutation from the vector space representation we extend the coordinate vector $\boldsymbol{p}$ as follow

$$(p_1 : v_1, \ldots, p_m : v_m), \text{ with } v_1, \ldots, v_m \in \Sigma^*.$$

Our goal is now to resort this list such that the resulting label sequence satisfies the order constraint given by the coordinates $\boldsymbol{p}$. To this end we perform an insertion sort like algorithm where we start with the rightmost element v_m since there is no element which can be larger, and p_m is therefore always zero. Furthermore let x be the image of the encoded permutation which we initialize with v_m. Subsequently, we concatenate the element v_{m-i}, $i = m-1, \ldots, 1$ with x_i and shift the inserted element v_{m-i} exactly p_{m-i} steps to the right, since the symbol v_{m-i} expects exactly p_{m-i} symbols on the left. We continue until we inserted all m^* elements, and the resulting sequence x_m is the image of the permutation. Let us reconsider the case for example E3.3

Example E3.4 From Vector Space back to the Permutation Group: *Let us decode the vector* $\boldsymbol{p} = (0, 0, 2, 2, 1, 0, 0)$*. We first assign each possible label in* Σ^* *to* p_i*:*

$$(0 : K, 0 : e, 2 : e, 2 : l, 1 : n, 0 : r, 0 : s).$$

Starting at the last entry and keeping concatenating and shifting we obtain the sequence:

$$\begin{aligned}
\textit{Start:} &\quad (s),\\
\textit{Insert (r) and shift 0 right:} &\quad (r, s),\\
\textit{Insert (n) and shift 1 right:} &\quad (r, n, s),\\
\textit{Insert (l) and shift 2 right:} &\quad (r, n, l, s),\\
\textit{Insert (e) and shift 2 right:} &\quad (r, n, e, l, s),\\
\textit{Insert (e) and shift 0 right:} &\quad (e, r, n, e, l, s),\\
\textit{Insert (K) and shift 0 right:} &\quad (K, e, r, n, e, l, s).
\end{aligned}$$

We are now ready to apply the CE Method to search for the right permutation. We propose to search in the vector space P, and before evaluating the loss function we have to decode our strings first. As sample generating device we propose to use the binomial probability function with

$$f(x; \boldsymbol{v}) = \prod_{j=1}^{m^*} \binom{j}{x_j} v_j^{x_j} (1 - v_j)^{j - x_j}.$$

Note that we only need to search over $m^* - 1$ coordinates since the last entry of the vector $\boldsymbol{p}$ is always zero. For brevity we are going to use the same symbols Z to denote the decoded sequence and the vector space representation of the necessary permutation. However, we will sample and optimize over permutations but use our decoding mechanism to calculate the loss $||\boldsymbol{\Psi} - \phi(Z_i)||^2$. Furthermore, to finally use the CE method we need to specify the

update equations for the optimal importance sampling distribution at step k. To this end we have to find the new parameter $\boldsymbol{v}^k$ such that

$$\boldsymbol{v}^k = \arg\max_{\boldsymbol{v}} \sum_{i=1}^{l} 1_{\{||\boldsymbol{\Psi}-\phi(Z_i)||^2 \leq \gamma\}} \ln f(Z_i; \boldsymbol{v}^{k-1}).$$

Thus the optimal update is given by

$$\hat{\boldsymbol{v}}_j = \frac{\sum_{i=1}^{l} 1_{\{||\boldsymbol{\Psi}-\phi(Z_i)||^2 \leq \gamma\}} Z_{ij}}{\sum_{i=1}^{l} 1_{\{||\boldsymbol{\Psi}-\phi(Z_i)||^2 \leq \gamma\}} j}. \tag{33}$$

To finally get the MAP update $\boldsymbol{v}^k$ we need to build our prior from possibly given example sequences in D_N first. Unfortunately this is not possible, since the representation is size dependent and assumes the same alphabet. We thus select the smoothing update $\boldsymbol{v}^k = (1-\alpha)\hat{\boldsymbol{v}} + \alpha \boldsymbol{v}^{k-1}$ with some α. This way we can still exploit the examples in D_N to create a reasonable start value $\boldsymbol{v}^0$. For example selecting the nearest neighbor gives us a reasonable initialization of $\boldsymbol{v}^0$. We summarize the pre-image algorithm for sequence reconstruction in algorithm A3.5. Finally, note that the stochastic search needs to evaluate the kernel function many times. Due to (30) this requires to calculate an inverse of a matrix for each kernel evaluation. For sequences, the involved adjacency matrices have structural constraints, and thus it is possible to exploit this structure to speed up kernel computation dramatically.[5]

In the next section we are going to drop our assumptions on the adjacency matrix and extend our stochastic search strategy for graphs.

3.4.5 Pre-images for labeled graphs

Since a sequence is a special case of a graph we can use the same techniques from sequences to determine size m^* and vertex set Σ^* of a graph $x = (\Sigma^*, E^*)$. Graphs with arbitrary adjacency matrix E require to decide for m^{*2} edges if they are part of the pre-image graph z^* or not. We are going to discuss mainly undirected graphs which imply a symmetric adjacency matrix and thus a smaller amount of edges to search.

To decide if an edge exists we propose to model every edge as a Bernoulli variable:

$$f(\boldsymbol{Z}; \boldsymbol{v}) = \prod_{1 \leq j < i \leq m^*} v_{ij}^{E_{\boldsymbol{Z}}(i,j)} (1 - v_{ij})^{1 - E_{\boldsymbol{Z}}(i,j)}.$$

Using the CE method we can simulate from the joint edge distribution function f and obtain an maximum likelihood estimate of the parameter $\boldsymbol{v}$ according to the update law

$$\boldsymbol{v}^k = \arg\max_{\boldsymbol{v}} \sum_{j=1}^{\frac{m^*(m^*-1)}{2}} 1_{\{||\boldsymbol{\Psi}-\phi(\boldsymbol{Z}_j)||^2 \leq \gamma\}} \ln f(\boldsymbol{Z}_j; \boldsymbol{v}^{k-1}). \tag{34}$$

[5] See chapter 4 in [81] for details.

Algorithm A3.5 SEQUENCE RECONSTRUCTION VIA CE METHOD

Given $\mathbf{\Psi}$, an example set D_N of sequences, the number l of candidate sequences to search and the maximum number of void iterations. The pre-image $z^* = (\Sigma_v^*, E^*)$ can be found by performing the following steps:

1. Determine the length m^* of the pre-image sequence z^* by for example (31).
2. Determine the occurrence frequency of symbol v_i appearing in the set Σ_v^* via (32).
3. Select a nearest neighbor $x_i \in D_N$ to calculate a start permutation vector $\boldsymbol{p}^0$ and thus a start value for $\boldsymbol{v}^0$.

   While $||\boldsymbol{v}^k - \boldsymbol{v}^{k-1}|| > \epsilon$ and an update happened in the last t iterations

   4. Generate a random sample $\{\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_l\}$ according the sampling distribution $f(\cdot; \boldsymbol{v}^{k-1})$.
   5. Determine the minimization level parameter γ^k.
   6. If $\gamma^k < \gamma^{k-1}$
   7. Determine the optimal importance sampling distribution parameters $\hat{\boldsymbol{v}}$ by solving (33).
   8. Update $\boldsymbol{v}^k = \alpha\hat{\boldsymbol{v}} + (1-\alpha)\boldsymbol{v}^{k-1}$
   9. k=k+1

The final z^* pre-image can be obtained by selecting the best example by sampling from $f(\cdot; \boldsymbol{v}^k)$.

Thus the best possible update according to the CE method follows by

$$\begin{aligned}
0 &= \frac{\partial}{\partial \boldsymbol{v}_{kl}} \frac{1}{N} \sum_{j=1}^{N} 1_{\{||\mathbf{\Psi}-\phi(\boldsymbol{Z}_j)||^2 \le \gamma\}} \ln f(\boldsymbol{Z}_j; \boldsymbol{v}), \\
0 &= \sum_{j=1}^{N} 1_{\{||\mathbf{\Psi}-\phi(\boldsymbol{Z}_j)||^2 \le \gamma\}} \frac{\partial}{\partial v_{kl}} \left(\sum_{i=1}^{d} \ln \left(v_{kl}^{E_{\boldsymbol{Z}_j}(k,l)} (1-v_{kl})^{1-E_{\boldsymbol{Z}_j}(k,l)} \right) \right), \\
0 &= \sum_{j=1}^{N} 1_{\{||\mathbf{\Psi}-\phi(\boldsymbol{Z}_j)||^2 \le \gamma\}} \left(\frac{E_{\boldsymbol{Z}_j}(k,l)}{v_{kl}} - \frac{1-E_{\boldsymbol{Z}_j}(k,l)}{1-v_{kl}} \right),
\end{aligned}$$

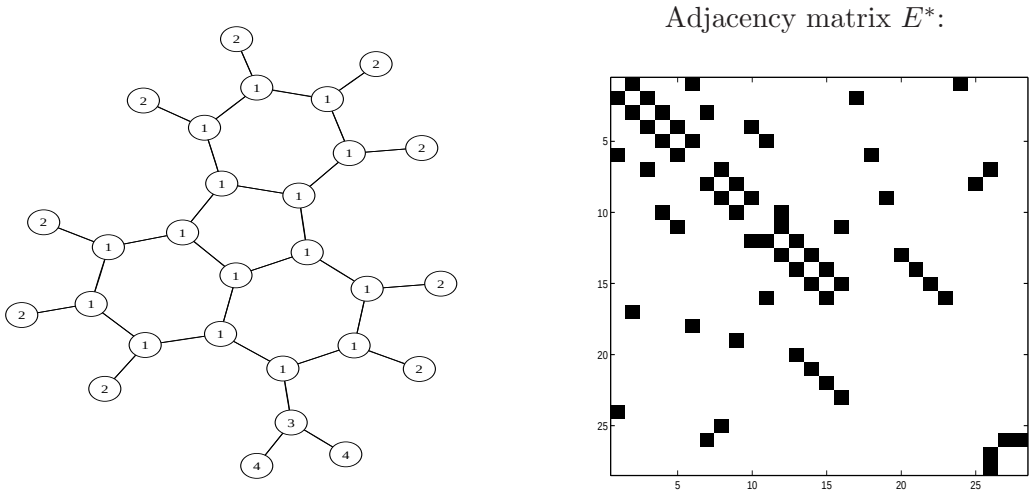
and thus

$$\boldsymbol{v}_{kl} = \frac{\sum_{j=1}^{N} 1_{\{||\mathbf{\Psi}-\phi(\boldsymbol{Z}_j)||^2 \le \gamma\}} E_{\boldsymbol{Z}_j}(k,l)}{\sum_{j=1}^{N} 1_{\{||\mathbf{\Psi}-\phi(\boldsymbol{Z}_j)||^2 \le \gamma\}}}. \tag{35}$$

To obtain a MAP estimate we consider an example graph in figure **??**, which represents a molecule used in the study of [19]. Obviously, the adjacency matrix of this graph is very sparse, thus we should penalize distributions which lead to non-sparse adjacency matrices. Thus we choose a Laplacian prior

$$P_2(\boldsymbol{v}|\lambda) = \prod_{i=1}^{\frac{m^*(m^*-1)}{2}} \frac{\lambda}{2} e^{-\lambda|v_i|},$$

where λ is a free parameter controlling the influence of the prior.



$$m^* = 28,\ \Sigma_v^* = \left\{16 \times \boxed{1}, 9 \times \boxed{2}, 1 \times \boxed{3}, 2 \times \boxed{4}\right\}$$

Figure 3.6: Two representations for an undirected graph. This graph is showing a molecule from the mutagen database, see [19].

Considering $P_1(F(Z)|\boldsymbol{v})$ to be Gaussian we can get a MAP update by

$$\begin{aligned} \boldsymbol{v}^k &= \arg\max_{\boldsymbol{v}} \ P_1(F(Z)|\boldsymbol{v})P_2(\boldsymbol{v}|\lambda) && (36)\\ &= \arg\min_{\boldsymbol{v}} \ (F(Z)-\boldsymbol{v})^2 + \lambda \boldsymbol{v}^\top \mathbf{1} && (37)\\ &\quad \text{subject to } \boldsymbol{v} \geq \mathbf{0}, && (38) \end{aligned}$$

where $\mathbf{1}$ denotes a vector of length $\frac{m^*(m^*-1)}{2}$ with all entries equal to one. Unfortunately driving components v_i to quickly to zero could harm the optimization, for this reason we allow a basic *driving* noise level ν, and thus modify the inequality constraint (38) to

$$\boldsymbol{v} \geq \nu \mathbf{1}. \tag{39}$$

Note that by adding a further Gaussian prior, which depends on the last estimate $\boldsymbol{v}^{k-1}$, we can integrate the typical smoothing update as well. Thus our final MAP estimate is given by

$$\begin{aligned} \boldsymbol{v}^k &= \arg\max_{\boldsymbol{v}} \ P_1(F(Z)|\boldsymbol{v})P_1(\boldsymbol{v}|\boldsymbol{v}^{k-1})P_2(\boldsymbol{v}|\lambda) && (40)\\ &= \arg\min_{\boldsymbol{v}} \ \frac{1}{2}(F(Z)-\boldsymbol{v})^2 + \frac{1}{2}(\boldsymbol{v}-\boldsymbol{v}^{k-1})^2 + \lambda \boldsymbol{v}^\top \mathbf{1} \\ &\quad \text{subject to } \boldsymbol{v} \geq \boldsymbol{\nu}. \end{aligned}$$

If we solve the unconstrained problem and project back to the feasible set afterwards, we can give a closed form solution by

$$\boldsymbol{v}^k = \max\left(\mathbf{1}\nu, \frac{1}{2}\left(F(Z) + \boldsymbol{v}^{k-1}\right) - \lambda \mathbf{1}\right), \tag{41}$$

where the max operation is component-wise. To initialize $\boldsymbol{v}$ we can use the nearest neighbor from D_N, analogously to the sequence pre-image case. A summary of the final pre-image algorithm is given in A3.6.

In the next section, let us investigate the introduced pre-image algorithm for sequences.

Algorithm A3.6 GRAPH RECONSTRUCTION VIA CE METHOD AND SPARSITY PRIOR.

Given $\mathbf{\Psi}$, an example set D_N of graphs, the number l of candidate graphs to search, the sparsity penalty λ, a drift noise parameter ν and the number t of maximal void iterations. The pre-image $z^* = (\Sigma_v^*, E^*)$ can be found by performing the following steps:

1. Determine the length m^* of the pre-image sequence z^* by for example (31).
2. Determine the occurrence frequency of symbol v_i appearing in the set Σ_v^* via (32).
3. Initialize proposal adjacency matrix E^* by nearest neighbor.

   While $||\boldsymbol{v}^k - \boldsymbol{v}^{k-1}|| > \epsilon$ and an update happened in the last t iterations

   4. Generate a random sample $\{\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_l\}$ according the sampling distribution $f(\cdot; \boldsymbol{v}^{k-1})$.
   5. Determine the minimization level parameter γ^k.
   6. If $\gamma^k < \gamma^{k-1}$
   7. Determine the optimal importance sampling distribution parameters $F(Z)$ by solving (35).
   8. Update $\boldsymbol{v}^k = \max\left(\mathbf{1}\nu, \frac{1}{2}\left(F(Z) + \boldsymbol{v}^{k-1}\right) - \lambda\mathbf{1}\right)$.
   9. k=k+1

The final z^* pre-image can be obtained by selecting the best example by sampling from $f(\cdot; \boldsymbol{v}^k)$.

3.5 Interpolating Sequences

Assume that we have given a set of sequence, and our goal is to construct new sequences in the vicinity of the given sequences. We could tackle this problem via our pre-image technique for sequences.

Let us define the linear combination of discrete objects by the linear combination of its coordinates given a feature space embedding ϕ_k. For example, to obtain the mean sequence z^* of all patterns we would need to calculate

$$z^* = \arg\min_{z \in \mathcal{Z}} ||| \frac{1}{N} \sum_{i=1}^{N} \boldsymbol{V}_r^\top \phi_k(x_i) - \boldsymbol{V}_r^\top \phi_k(z)||^2,$$

given some orthogonal basis $\boldsymbol{V}_r$ of the subspace spanned in feature space $\mathcal{F}_k$ by all points in $D_N = \{x_i, y_i\}_{i=1}^N$. In this subsection, we want to investigate if it is possible to *interpolate* between sequences x_1 and x_2. Thus, we seek to find a pre-image according to

$$z^* = \arg\min_{z \in \mathcal{Z}} |||(1-\alpha)\boldsymbol{V}_r^\top \phi(x_1) + \alpha \boldsymbol{V}_r^\top \phi(x_2) - \boldsymbol{V}_r^\top \phi_k(z)||^2, \text{ for } \alpha \in [0,\, 0.1, \ldots, 1].$$

To this end, let us introduce two stochastic automata $\mathsf{A}_1 = (\pi_1, T_1)$, $\mathsf{A}_2 = (\pi_2, T_2)$ with five states each and different transition probability matrices T_1, T_2 and different start probabilities π_1, π_2. We initialize both transition matrices randomly, where we have selectively removed some entries and ensure that each row sum to 1 (see figure 3.7). By sampling from these automata we generate 250 sequences of length $20 \leq l \leq 30$ from each automata and calculate the nonlinear principal components using the marginalized kernel and stopping probability $\frac{1}{l}$.

Let us now pick randomly the two sequences x_1, x_2 (shown in figure 3.9) which are generated from the automata $\mathsf{A}_1, \mathsf{A}_2$ respectively. We use algorithm A3.5 with various size of candidate sequences l and various maximum iterations. We compare the relative

22555234455555244455 5

44555555555553333444445555

1245555245555555334

333455555555555555555525344555

33333445555252225553

444555555213333344444445555555

33334444455555552252345555533

123333444444444534444445525 5

11255555555333333333333344

444533334444444555555244553

14454425554445525555

25555235554455555555

12555555555555442555

155554255555555552555555555 2

23333442355555234444

11555544255425555235

23342555555542312355

14255555544233425523

125555423522352523444445444 4

23334444255555555555

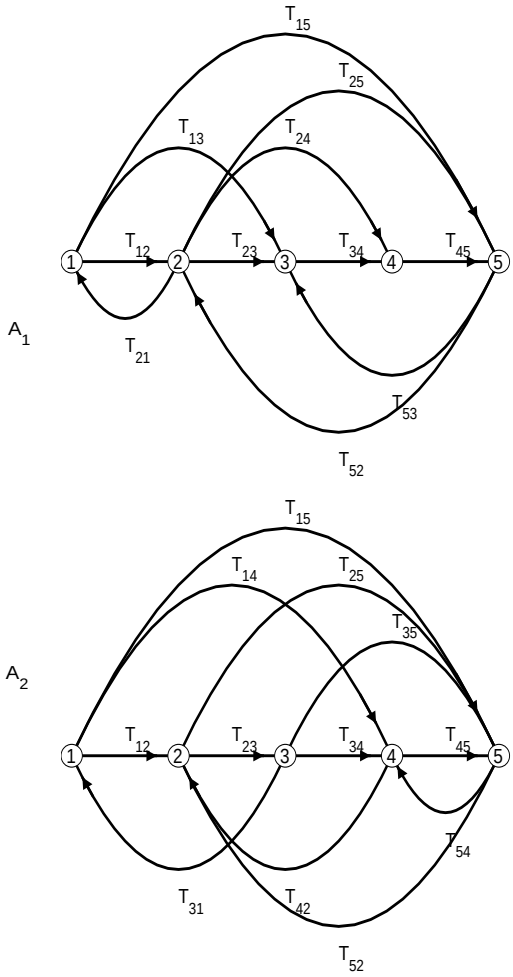


Figure 3.7: Used automata and typical sequence sampled from automata.

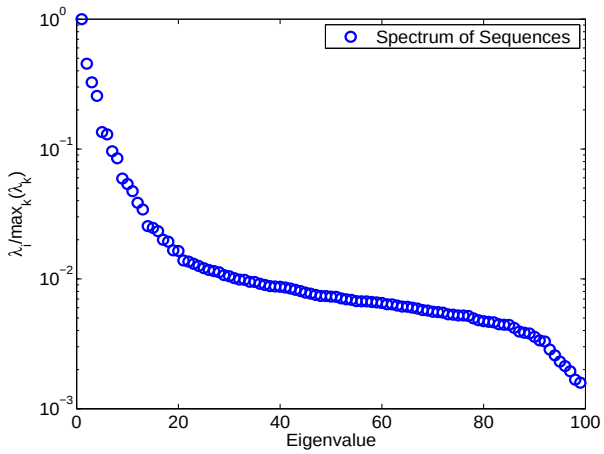


Figure 3.8: Using the marginalized kernel, 90 % of the variance is retained in the first ten principal directions where we have use the first two for plotting paths.

distance of our pre-image sequence with the nearest neighbor from the sampled data, where we use the feature space distance of the nearest neighbor as normalization constant. The found sequences are shown in table 3.9. In figure 3.10 we plot the first two principal

α	Sequence	
0.0	21123345525555334555555524	x_0
0.1	2223312553345545555555	
0.2	2312322553345554555555	
0.3	2122532533455354555555	
0.4	2152512533455554554555	
0.5	12524252535545555555	
0.6	12252552455545555555	
0.7	125252245254555555555	
0.8	125522525555455555555	
0.9	122524555255555555555	
1.0	12555552555555252555	x_1

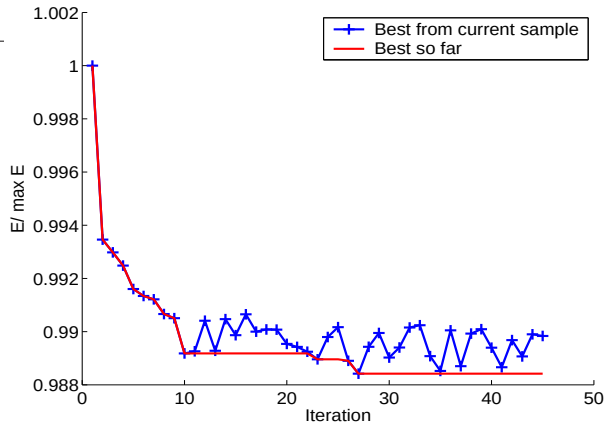


Figure 3.9: Left: Found pre-image sequences for $\alpha = 0, \ldots, 1$. Right: Sampling and iterative refinement of distribution while using the CE method for sequence reconstruction.

coordinates of all sequences and the relative distance of synthesized sequences. Note that the found pre-image is better than the nearest neighbor in regions where the data is sparse, which is in our case in the vicinity of the mean.

3.6 Conclusion

In this chapter we were concerned with the pre-image problem in kernel methods. Thus we have considered the problem of reconstructing patterns from a feature map. We have introduced techniques to reconstruct corresponding patterns in the input space which avoid difficult and/or unstable numerical optimization and are easy to implement. Compared to classical approaches, the new methods have the advantage that they are numerically stable, are much faster to evaluate and better suited for high-dimensional input spaces. Algorithms as KDE can benefit from this fact, since they need to solve the pre-image problem for each prediction. We have introduced one pre-image technique based on regression which is 3 orders faster in evaluation than the classical technique solely based on ptimization while yielding better results.

Pre-image techniques were introduced when the input space is discrete and gradients do not exist. We have explored pre-image algorithms for structures like graphs and sequences and have incorporated a-priori knowledge in the pre-image learning stage which is crucial for discrete pre-images. We demonstrated the algorithm for sequence pre-images by constructing the *mean* of sequences.

The pre-image problem is an ill-posed problem. Therefore, we conclude that reconstructing the pre-image can benefit substantially from any additional information one can provide to the reconstruction algorithm. As was noted in the introduction, since the pre-image problem is the problem dependent front end of KDE, each application will require its own special pre-image technique. This chapter provides an arsenal of methods that can be used for this purpose.

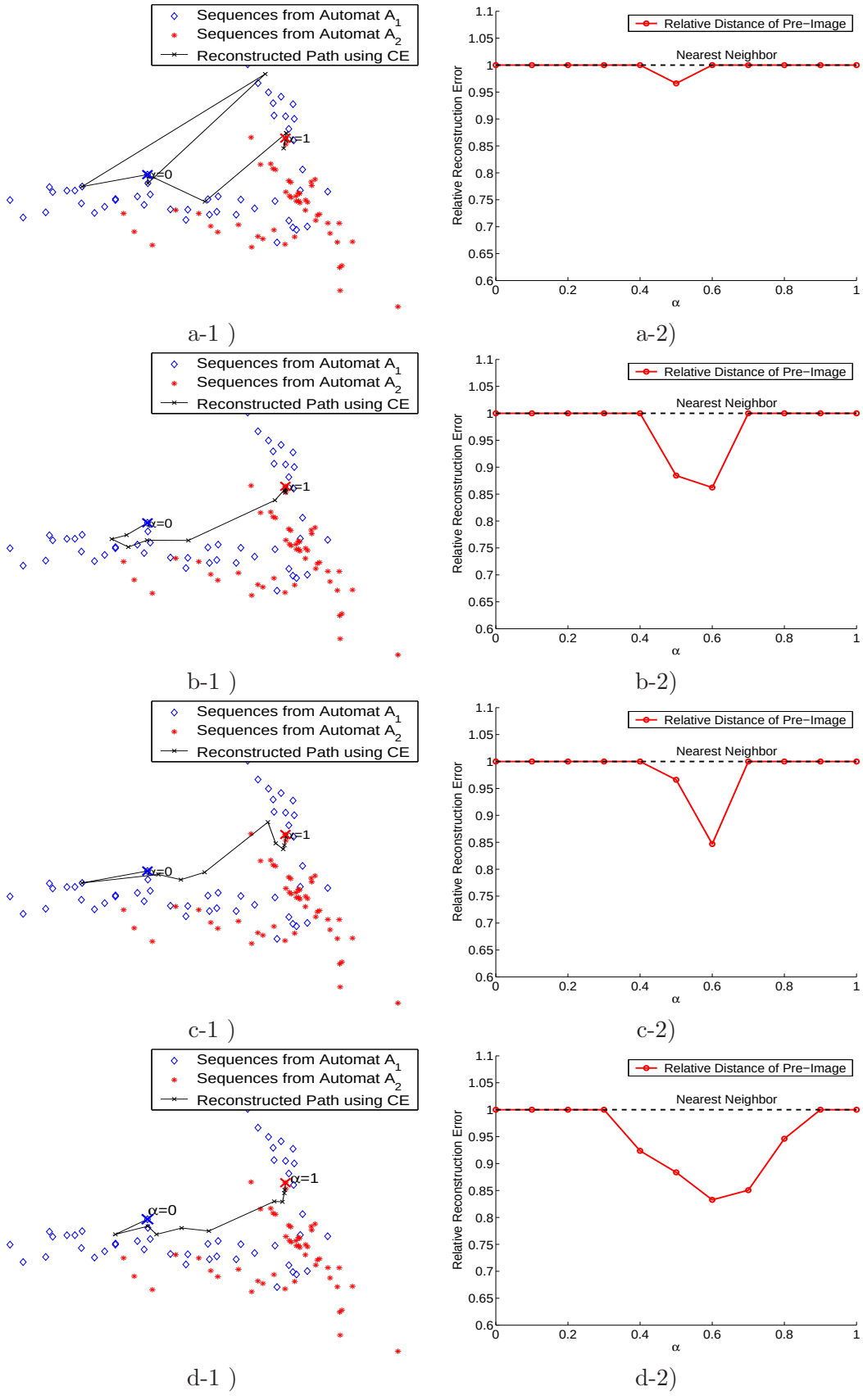


Figure 3.10: Synthesized interpolations for various experimental settings projected onto the first two eigenvectors. Original data with start ($\alpha = 0$) and end ($\alpha = 1$) points of interpolation are marked. a) Using 5 samples and 5 failed iterations. b) Using 20 samples and 5 failed iterations. c) Using 20 samples and 20 failed iterations. d) Using 200 samples and 20 failed iterations. The found pre-image sequences are the better the longer we search and the more samples we use for sampling.

Chapter 4

Speeding up KDE by Reduced Set Methods.

Geschwind gewinnt.
– German aphorism –

Kernel Dependency Estimation uses various kernel algorithms which all require extensive calculation of kernel functions during prediction. For time-critical applications such as robotics this is a serious drawback. In this chapter we investigate post-processing techniques aiming at speeding up the kernel expansions involved in KDE.

4.1 Introduction

In the introductory chapter 1 it was mentioned that kernel algorithms are mostly based on the representer theorem (see T1.3, page 12) which guarantees that the solution of the optimization problem

$$w^* = \arg\min_f \sum_{i=1}^{N} \ell(y_i, x_i, w^\top \phi_k(x_i)) + \Omega[w]$$

is in the span of the mapped data, i.e. $w^* = \sum_{i=1}^{N} \alpha_i \phi_k(x_i)$. Here, x, y are the training data $D_N = \{x_i, y_i\}_{i=1}^N$ and $\phi_k : \mathcal{X} \to \mathcal{F}_k$ corresponds to the feature map implied by the kernel function k. As already mentioned in the introductory chapter, the representation in terms of kernel expansions is also called the *dual* representation.

Obviously, the computational complexity of the prediction equation

$$w^\top \phi_k(x) = \left(\sum_{i=1}^{N} \alpha_i \phi_k(x_i) \right)^\top \phi_k(x) = \sum_{i=1}^{N} \alpha_i k(x_i, x) \tag{1}$$

depends on N, the size of the expansion necessary to express w. Furthermore it depends on the time it takes to evaluate the kernel function k itself. Let us assume that the kernel function is *atomic*, thus we will ignore its time complexity and our interest is to reduce the number of kernel evaluations only. [1] Therefore, the *sparsity* of the dual representation (1), i.e. the number of coefficients α in (1) not zero, is of great interest since it determines

[1] For an approach concerned with speeding up the kernel function, see [48].

the number of necessary kernel computations and thus effectively the prediction speed which is our main concern in this chapter. For this reason, we will investigate so-called *reduced set* techniques, which try to reduce the number of kernel evaluations in the dual representation and thus aim at increasing prediction speed. Essentially, such methods might become useful in one of the following scenarios:

(i) if the predictor one wishes to compress is already sparse, but the basis it chooses still has linearly dependent or *close to* linearly dependent examples (e.g., in SVMs in the presence of *noise* [2, 87]),

(ii) if the predictor always takes expansions in the full dataset as in, e.g., the typically used methods in KDE such as Ridge Regression or Kernel PCA,

(iii) when one wishes to construct a multi-output predictor where each predictor is sparse but there is a redundancy due to independent learning of multiple outputs.

In particular we will focus on *reduced set selection*(RSS)[2] techniques which we introduce in the next section. Before doing so, let us discuss in the following why it is possible to drop examples from the expansion in (1). We present two reasons which give rise to the two classes of algorithms in the literature.

Linear Dependence Consider the case that $\mathcal{X}$ is a real vector space, e.g. $\mathbb{R}^d$, and we are using a *linear* kernel function corresponding to the identity map as feature map ϕ_k. In this case, whenever we have more points N than dimensions in feature space, the set of points $\{x_i\}_{i=1}^N$ cannot constitute a linearly independent family. It is clear that in this case there must exist a smaller expansion, i.e

$$N > \dim \mathcal{F}_k \Rightarrow \exists \sum_{i=1}^{\dim \mathcal{F}_k} \beta_i \phi_k(x_{I(i)}) = \sum_{i=1}^{N} \alpha_i \phi_k(x_i),$$

where β are new coefficients and $I \subset \{1, \ldots, N\}$ is an index set choosing a linear independent subset of points of the training data D_N. This is not very exciting so far since our argumentation was based on the assumption that we use a linear kernel. Interestingly, the situation does not change essentially when we use a non-linear feature map. Although the dimensionality of $\mathcal{F}_k$ is much higher than N, the number of points in the high dimensional subspace are likely to span a space which is of *smaller* dimension than N. In the special case that the feature map of kernel k corresponds to a high but finite dimensional feature space, this is surely the case for N tending to infinity. However in practice, even for kernels which lead to an infinite-dimensional feature space such as the Gaussian kernel, one will yield almost linearly dependent feature vectors due to precision effects, see [87]. We will demonstrate this effect in the experimental section.

Irrelevant span Consider the case of classification and that the hyperplane normal lives in a subspace of all N points. In such a situation, although the N points in the dataset span an N dimensional feature space, the hyperplane can be expressed by less points. Therefore, the linear independence of the points is not relevant for the task at hand. Note that from this fact we can formulate the selection task as: *Drop points as*

[2]Note that a related, but more difficult task is that of reduced set *construction*. In that task one searches for weights β_i and vectors $\phi_k(z)_i$, where $\phi_k(z)_i$ are not necessarily in the training set. This results in algorithms which give higher compression rates, but are much slower and more difficult to optimize, see e.g. [13].

long as it does not harm the classification rule. We will see later how we can formulate a concrete optimization problem which tries to solve this task.

Due to the presented reasons, the same point w in the feature space $\mathcal{F}_k$ can be expressed by multiple linear combinations of $\phi_k(x_i)$ almost surely for the typically used kernel functions. Note that this is in contrast to the convexity property of the involved optimization problem in (T1.3), e.g. in SVMs the solution w is unique, but may have multiple representations in the dual space. In the following sections, we review existing techniques for RSS which try to solve one of the tasks: removing linear dependent points or trying to drop irrelevant points. Afterwards we propose new methods for the concrete case of classification $\mathcal{Y} = \{+1, -1\}$ and regression $\mathcal{Y} = \mathbb{R}$. We generalize RSS to the multiple-output case $\mathcal{Y} = \{1, \ldots, d\}$ and $\mathcal{Y} = \mathbb{R}^d$ with $d > 1$. Compared to existing methods, the methods proposed are faster to compute, yield similar or higher compression rates and are especially efficient in the multi-output case. Finally we validate the introduced algorithms experimentally on several classification and regression benchmarks, highlighting when RSS proves to be useful.

4.2 Existing reduced set selection techniques

In this section, we review existing techniques for reduced set selection techniques. In particular, we follow [78] and start with techniques trying to remove linear dependent points from expansion (1).

Approximations based on rank deficiency It can be shown that linear dependence between $D_N = \{x_i\}_{i=1}^N$ in feature space results in a rank deficient kernel matrix.

Theorem T4.1 Rank and Linear Dependence
The maximum linear independent subset $E_{N_2} \subset D_N$ with N_2 elements limits the rank of K to N_2, where K is the matrix of the inner products $K_{ij} = \phi(x_i)^\top \phi(x_j) = k(x_i, x_j)$.

Proof. We give a constructive proof. Write K in its spectral form $K = \sum_{i=1} \lambda_{i=1}^N v_i v_i^\top$. Obviously rank K equals the number of nonzero eigenvalues λ. Furthermore the eigenvectors of the empirical correlation matrix $\sum_{i=1}^N \phi(x)\phi(x)^\top$ can be expanded in the basis $\{v_1, \ldots, v_N\}$ formed by the eigenvectors of K,i.e. $z_i = \sum_{j=1}^N v_{ij}\phi(x_j)$. Therefore, an eigenvalue larger then zero means that there must be at least one point $x \in D_N$ with $\phi(x)^\top z_i > 0$. For all nonzero eigenvalues λ_i select one point x with $\phi(x)^\top z_i > 0 = \sum_{j=1}^N v_{ij} x^\top x_i > 0$. We can only select $r = \text{rank } K$ points and since $z_i^\top z_j = 0$ for all $i \neq j$ we have selected r linearly independent points. Since the remaining eigenvalues are zero, any further point must be in the span of the selected points since it is in the span of the eigenvectors z_i. □

This property was used in Downs et al [21] (see also [79]) to select a subset of size rank K. The subset was selected by computing the row echelon form of K and discarding points which lead to zero rows. Once the subset $Z = \{z_i\}_{i=1}^{N_2} \subset D_N$ is chosen, the expansion coefficients β can be calculated as

$$\beta = K_{Z,Z}^{-1} K_{Z,X} \alpha, \tag{2}$$

with

$$K_{Z,Z_{ij}} = k(z_i, z_j), \text{ and } K_{Z,X_{ij}} = k(z_i, x_j).$$

The problem with rank based techniques is their computational cost. The rank is very expensive to calculate for bigger samples since the computation complexity is $O(N^3)$ (row

echelon form, eigenvector decomposition, etc.). Furthermore, the compression rates may not be as one would like since this is a lossless technique, e.g. for RBF kernels which lead to kernel matrices of full rank (see [60]) no compression is possible[3]. This is a general property of RSS methods aiming at identifying linearly dependent points. Alternatively, one might sacrifice some accuracy in favor of sparsity which leads to the so-called lossy compression schemes. The next method is based on this idea.

Using an additional penalizer The ℓ_1 penalization method suggested in [78, sect. 18.4.2] simply attempts to construct a sufficiently good approximation of w by solving the following optimization problem

$$\arg\min_{\beta} \left\| w - \sum_i \beta_i \phi_k(x)_i \right\|^2 + \gamma \sum_i c_i |\beta_i| \tag{3}$$

where parameter γ determines the trade-off between accuracy and sparsity. The first term in (3) tries to *represent the original model w as good as possible* whereas the second term *keeps the coefficients as sparse as possible*. The weights c_i specify a prior on the importance of reducing the weight of example i. Choosing $c_i = 1$, $i = 1, \ldots, N$ would lead to a uniform prior on dropping a point. Expression (3) yields a numerically tractable quadratic programming problem. However, if all patterns are used ($\forall i : c_i = 1$), this formulation has the disadvantage to be of cubic complexity in the number of patterns, i.e. $O(N^3)$, which might be too expensive. For the special case of SVM, one can follow [79] which suggested to take $c_i = 1/\alpha_i$ in order put more emphasis on reducing smaller weights. In that case, one can simply ignore the points with $\alpha_i = 0$ which for the SVM method gives a complexity cubic in the number of support vectors, i.e. $O(\#SV^3)$. However, for kPCA or Ridge Regression which are not sparse, this does not hold and there is no prior regarding the expansion.

Analog to the single-output case, the concept of additional penalizers can be used for a multi-output/class scenario which leads as well to a quadratic optimization problem:

$$\arg\min_{\beta^{1\pm},\ldots,\beta^{p\pm}\in\mathbb{R}^{2N},\xi\in\mathbb{R}^N} \sum_j \left\| w^j - \sum_i (\beta_i^{j+} - \beta_i^{j-}) \phi_k(x)_i \right\|^2 + \gamma \sum_i c_i \xi_i \tag{4}$$

subject to

$$\beta_i^{j+} + \beta_i^{j-} \leq \xi_i, \quad \xi_i, \beta_i^{j+}, \beta_i^{j-} \geq 0.$$

Here ξ_i are variables which measure the weight of a sample x_i for each of the q outputs. Note that the solution of this optimization problem has complexity $O((Nq)^3)$ where N is number of examples and q is number of outputs. Thus solving this problem does not scale well with the number of output dimensions involved. For example, if we consider the case of kPCA and if we would like to keep all principal directions the complexity of solving (4) would be $O(N^4)$. Another drawback of this method is that controlling the resulting approximation error $\|w - \sum_i \beta_i \phi_k(x)_i\|^2$ by adjusting γ seems to be quite difficult in practice.

4.3 Fast Reduced Set Selection

In this section we describe two novel algorithms for achieving faster reduced set selection: hyperplane matching pursuit, and minimization of the so-called zero-norm. In particular,

[3] Apart from finite precision effects on computers. This may explain why [21] reported a reduction in SVs for RBF kernels.

we focus on generalizations of these methods that are especially efficient in the multi-output case. We start with the application of a classical algorithm known as matching pursuit to reduced set selection. Afterwards we extend it to the case that we do not want to compress only a single but multiple linear models $\{w^1, \ldots, w^p\}$ at once. As an alternative to matching pursuit, we investigate the usage of the so-called *zero-norm* minimization for reduced set selection. Let us begin with the matching pursuit algorithm.

4.3.1 Hyperplane Matching Pursuit

Matching pursuit (MP) has been used before in the machine learning community to greedily train kernel classifiers by [96]. The general approach of matching pursuit works by adding one training example on each iteration, trying to improve a given error measure, e.g. classification or regression loss. In our application we will consider the error measure $||w - \hat{w}||^2$ where $\hat{w}$ is our approximation which is given by $\hat{w} = \sum_{i=1}^{r} \beta_i \phi_k(z_i)$ and w is the original expansion (e.g. the hyperplane normal in SVM or an eigenvector expansion from kPCA). Note that the error measure can be expressed using kernel functions only. MP starts with the empty model $\hat{w} = 0$ and is given at iteration $i > 1$ the model w and a current set $Z = \{z_1, \ldots, z_{i-1}\}$ of samples used for the new approximation $\hat{w}$. For each iteration r, it tries to identify the optimal yet unused sample and its expansion coefficient β_r by minimizing the error measure, i.e.

$$(\beta_r, z_r) = \underset{z \in D_N \setminus Z,\ \beta \in \mathbb{R}}{\arg\min} ||w - \sum_{i=1}^{r-1} \beta_i \phi_k(z_i) - \beta \phi_k(z)||^2. \tag{5}$$

Note that we consider the set of $Z = \{z_i\}_{i=1}^{r}$ to be a subset of the originally given training data D_N, in contrast to reduced set construction. With increasing r, the error must decrease and therefore the approximation has to become better. However, we can simplify (5) by decomposing the minimization problem into two simpler problems that are faster to solve. Instead of minimizing the distance over the new expansion coefficient and the new sample index at the same time, we can also identify the pattern being as collinear in a first step, i.e:

$$z_r = \underset{z \in D_N \setminus Z}{\arg\min} \angle\left(w, \phi_k(z)\right) = \underset{z \in D_N \setminus Z}{\arg\max} \left| \frac{w^\top \phi_k(z)}{w^\top w \cdot \phi_k(z)^\top \phi_k(z)} \right|. \tag{6}$$

Afterwards, in a second step, we pick the optimal expansion coefficient by:

$$\beta_r = \underset{\beta \in \mathbb{R}}{\arg\min} ||w - \sum_{i=1}^{r-1} \beta_i \phi_k(z_i) - \beta \phi_k(z_r)||^2. \tag{7}$$

Note that all operations can be expressed with kernel functions only and thus it is not required to operate in feature space directly. We summarize this algorithm in A4.1. In step (2) the example is added to the reduced set which results in the largest decrease in $w^i = ||w - \hat{w}||^2$ in a greedy forward selection manner. In step (3) the resulting multiplier β_i is set. This process can be repeated until either one achieves a certain tolerance $||w - \hat{w}||^2 < \epsilon$ or alternatively one could measure the error rate (which is what one is really interested in) on a validation set. We now discuss possible improvements of the basic algorithm.

Backfitting The decrease in $||w - w^i||$ at each iteration can be improved considerably by optimally adjusting all β_i in the currently selected examples, rather than just the

Algorithm A4.1 MATCHING PURSUIT ON W

Given stopping criteria ϵ, a kernel matrix K, $w = \sum_i \alpha_i \phi_k(x)_i$ and training points $D_N = \{x_i, y_i\}_{i=1}^N$

Initialization

1. i=1, $\hat{w}^i = 0$

While $||w - \hat{w}^i||^2 > \epsilon$

2. Pick best sample z_i by solving (6)
3. Pick best coefficient β_i by solving (7)
4. Set new approximation $\hat{w}^{i+1} = \hat{w}^i + \beta_i \phi_k(z_i)$
5. $i = i + 1$.

Final approximation of w is given by $\hat{w} = \sum_j \beta_j \phi_k(x_j)$.

new incoming example. This approach is called backfitting, which is a standard trick in adaptive filtering [76] and was also used for kernel matching pursuit by [96]. To do this one solves

$$\boldsymbol{\beta} = \arg\min_{\beta \in \mathbb{R}^r} ||w - \sum_{i=1}^{r-1} \beta_i \phi_k(x_i) - \beta_r \phi_k(x_r)||^2. \tag{8}$$

Thus we re-optimize all previous found expansion coefficients. Note that, given $w = \sum_{i=1}^N \alpha_i \phi_k(x_i)$, we can obtain $\boldsymbol{\beta}$ analytically as as

$$\boldsymbol{\beta} = K_{Z,Z}^{-1} K_{Z,X} \alpha. \tag{9}$$

Here, $K_{Z,Z}$ denotes the kernel matrix between examples in Z (these are picked by matching pursuit), and $K_{Z,X}$ is the kernel matrix between all patterns in Z and all patterns used to express w. Obviously, performing backfitting has a higher computational cost per iteration than step (3) of the basic algorithm A4.1. However, note that backfitting costs $O(|Z|^3)$ for a single matrix inversion and is still more efficient than ℓ_1-minimization in section 4.2 which has complexity $O(N^3)$ since usually $|Z| = r \ll N$.

To improve the speed of backfitting, we can use the fact that the expansion is extended at each iteration by a single new sample z_r and therefore the matrix $K_{Z,Z}$ has the block-matrix structure

$$K_{Z,Z} = \begin{pmatrix} K_{Z\setminus\{z_r\},Z\setminus\{z_r\}} & K_{Z,Z\setminus\{z_r\}} \\ K_{Z,Z\setminus\{z_r\}}^\top & K_{\{z_r\},\{z_r\}} \end{pmatrix}.$$

We can use the inverse obtained in a previous iteration to quickly construct the new inverse. A straight forward way to implement this is to perform a Cholesky factorization and to apply the matrix inversion lemma. Before doing so, let us rename for convenience our quantities by $K_r = K_{Z,Z}$, $K_s = K_{Z\setminus\{z_r\},Z\setminus\{z_r\}}$, $\boldsymbol{k}_{r\setminus s} = K_{Z,Z\setminus\{z_r\}}$ and $k_{rr} = K_{\{z_r\},\{z_r\}}$. We can rewrite the kernel matrix K_r evaluated after iteration r as

$$K_r = \begin{pmatrix} K_s & \boldsymbol{k}_{r\setminus s} \\ \boldsymbol{k}_{r\setminus s}^\top & k_{rr} \end{pmatrix}.$$

Now, the inverse after iteration r can be written as as

$$K_r^{-1} = \begin{pmatrix} K_s & \boldsymbol{k}_{r\setminus s} \\ \boldsymbol{k}_{r\setminus s}^\top & k_r \end{pmatrix}^{-1} \tag{10}$$

$$= \begin{pmatrix} K_s^{-1} + \lambda K_s^{-1}\boldsymbol{k}_{r\setminus s}\boldsymbol{k}_{r\setminus s}^\top K_s^{-1} & \lambda K_s^{-1}\boldsymbol{k}_{r\setminus s} \\ \left(\lambda K_s^{-1}\boldsymbol{k}_{r\setminus s}\right)^\top & \lambda \end{pmatrix}, \tag{11}$$

with $\lambda = \frac{1}{k_{r,r} - \boldsymbol{k}_{r\setminus s}^\top K_s^{-1}\boldsymbol{k}_{r\setminus s}}$. We apply now the Cholesky factorization and obtain $K_r^{-1} = R_r^\top R_r$, which leads to the final recursion equation for the matrix inverse of K_r:

$$R_r = \begin{pmatrix} R_s & \mathbf{0} \\ -\sqrt{\lambda} R_s^\top R_s \boldsymbol{k}_{r\setminus s} & \sqrt{\lambda} \end{pmatrix}. \tag{12}$$

In general we found out that backfitting is a critical step in MP style algorithms and that using MP without backfitting is by far worse.

Optimizing the real-valued output We currently minimize $||w - \hat{w}||^2$. However, we are not so much interested in this difference as in the difference in prediction of the two rules. From the Cauchy-Schwartz equation the latter is only an upper bound to the generalization error, namely:

$$e_x = \mathbb{E}_x ||w^\top \phi_k(x) - \hat{w}^\top \phi_k(x)||^2 \leq \sup_x \; k(x,x)||w - \hat{w}||^2.$$

While we try to minimize the right hand side, we could try to minimize the left hand side by the empirical estimate of e_x, namely $\hat{e}_x = \frac{1}{N}\sum_{i=1}^N ||w^\top \phi_k(x)_i - \hat{w}^\top \phi_k(x)_i||^2$ which is the squared difference of the predictions on the training set. Interestingly, the β which we obtain by minimizing (8) are equivalent to the β which minimize:

$$\frac{1}{|Z|}\sum_{i=1}^{|Z|} ||w^\top \phi(z_i) - \hat{w}^\top \phi(z_i)||^2 \tag{13}$$

as the minimizer of (13) is also (9). This shows that the $||w - \hat{w}||^2$ minimizer is suboptimal in terms of prediction as it only minimizes the difference in prediction on a *subset* of the examples. Thus in principle it is always possible to improve backfitting by using a validation set.

Using Cholesky factorization and matching pursuit we have obtained an efficient reduced set selection algorithm. Let us now extend its applicability from compression of a single model to the compression of multiple models.

4.3.2 Multiple-Hyperplane Matching Pursuit

In multi-class classification or multivariate regression one typically finds a solution which is a combination of several linear models, e.g. in the one-vs-the-rest or one-against-one approaches [22] of classification one obtains a *set* of classifiers. For example in one-vs-the-rest classification one trains p classifiers for p classes, where the j-th hyperplane w^j is learned with examples labeled positively if they are in class j and negative otherwise. This gives a single final meta-classifier:

$$f(x) = \arg\max_{1\leq j\leq p} ({w^j}^\top \phi_k(x))$$

where each w^j is given again by an expansion of data points, i.e.:

$$w^j = \sum_{i=1}^{N} \alpha_i^j \phi_k(x_i).$$

If we use use the standard simple model reduced set methods, we could compress each hyperplane *independently*. However, the main computational cost we are trying to reduce is in the calculation of the overall number of evaluated kernel functions. Thus we would rather minimize the union of expansion vectors $|\{i : \sum_{j=1}^{p} |\alpha_i^j| > 0\}|$ than the sum: $\sum_{j=1}^{p} |\{i : |\alpha_i^j| > 0\}|$. We thus wish to couple the compression steps to compress all the hyperplanes $\{w^1, \ldots, w^p\}$ all at once. We can do this in the matching pursuit algorithm by choosing the next sample z_r as the best sample which minimizes *all* reconstruction errors together, i.e.:

$$z_r = \arg\min_{z_r \in D_N \setminus Z} \min_{\beta_r^1, \ldots, \beta_r^p} \sum_{j=1}^{p} ||w^j - \sum_{i=1}^{r-1} \beta_i^j \phi_k(z_i) - \beta_r^j \phi_k(z_r)||^2$$

Analog to MP for the single model compression case, we can break this problem into parts and pick the optimal sample z_r and expansion coefficients one after the other. Thus, to perform Multiple Hyperplane Pursuit, we only need to replace the error measure in Algorithm A4.1.

One can argue that MP performs sub-optimally since it is a greedy approach. To this end, in the next section we will try to improve the existing penalty based approaches by considering a penalty term that enforces sparsity in the strictest possible way: The Zero Norm.

4.3.3 ℓ_0-norm Reduced Set Selection

Our central goal is to approximate the vector α of our original model w with a new sparse vector β, ideally *minimizing the number of nonzero coefficients of* β. The pseudo-norm which implements this is the co called zero-norm:

$$||\boldsymbol{\beta}||_0 = |\{\beta_i \neq 0\}|,$$

i.e. the zero norm counts the number of non-zero elements in the vector $\boldsymbol{\beta}$. Thus, ideally we would like:

$$\boldsymbol{\beta} = \arg\min_{\boldsymbol{\beta}} ||\boldsymbol{\beta}||_0 \tag{14}$$

subject to

$$\left|\left| \sum_{i=1}^{N} \alpha_i \phi_k(x_i) - \sum_{i=1}^{N} \beta_i \phi_k(x_i) \right|\right|^2 = 0 \tag{15}$$

Use of the so-called zero-norm has been researched before in the field of feature selection by [58, 98]. In [98] it was shown that the above problem is related to the problem:

$$\boldsymbol{\beta} = \arg\min_{\boldsymbol{\beta}} \sum_{i=1}^{N} \ln(\epsilon + |\beta_i|) \tag{16}$$

subject to (15), where the $\epsilon > 0$ is sufficiently small that it can be ignored when $\beta_i > 0$ but large enough to keep ln from going to $-\infty$ if β_i equals zero. This new objective

function is shown to be an upper bound of the zero norm. It has the advantage of being smooth which implies that the β_i can be computed by standard methods for continuous optimization, e.g. gradient descent. Instead of solving (16) by a plain gradient descent type algorithm we we will follow [98] and derive an iterative algorithm based on sequential linear programming and multiplicative updates. Before introducing the details, let us reformulate the objective function in (16) to get rid of the absolute value. We introduce new optimization variables and add additional constraints such that the absolute value will be automatically implemented as follows:

$$\begin{aligned} \boldsymbol{\beta}, \boldsymbol{v} &= \arg\min_{\boldsymbol{\beta},\boldsymbol{v}} \sum_{i=1}^{N} \ln(\epsilon + v_i) && (17) \\ \text{subject to} & \\ & \left\| \sum_{i=1}^{N} \alpha_i \phi_k(x_i) - \sum_{i=1}^{N} \beta_i \phi_k(x_i) \right\|^2 = 0 && (18) \\ \text{and} \quad & -\beta_i \leq v_i \leq \beta_i. && (19) \end{aligned}$$

Our goal is to solve convex programs in the vicinity of the current expansion vector $\boldsymbol{\beta}^k$ that identify a special direction of descent for the variables $\boldsymbol{\beta}$ and $\boldsymbol{v}$. If we would choose the direction of *steepest* descent we would need to solve a non-convex optimization line-search problem. In contrast, we will construct our descending directions in such a way such that this can be avoided. Let us first denote by $l(\boldsymbol{v})$ the objective function in (17). We start by considering the vicinity of l at a point $\boldsymbol{v}^k$ and make a first order approximation that yields:

$$f(\boldsymbol{v}) = l(\boldsymbol{v}^k) + \nabla l(\boldsymbol{v}^k)^\top \left(\boldsymbol{v} - \boldsymbol{v}^k \right),$$

where $\boldsymbol{v}^k$ is our reference vector in $\mathbb{R}_+^N$ resulting from previous iterations. The gradient can be easily identified as $\nabla l(\boldsymbol{v}^k) = \sum_{i=1}^{N} \frac{1}{v_i^k}$. Therefore, the new optimization problem is

$$\begin{aligned} \boldsymbol{\beta}, \boldsymbol{v} &= \arg\min_{\boldsymbol{\beta},\boldsymbol{v}} l(\boldsymbol{v}) = \arg\min_{\boldsymbol{\beta},\boldsymbol{v}} \sum_{i=1}^{N} \frac{v_i}{v_i^k} && (20) \\ \text{subject to} & \\ & \left\| \sum_{i=1}^{N} \alpha_i \phi_k(x_i) - \sum_{i=1}^{N} \beta_i \phi_k(x_i) \right\|^2 = 0 && (21) \\ \text{and} \quad & -\beta_i \leq v_i \leq \beta_i. && (22) \end{aligned}$$

We can now re-parametrize the optimization problem by replacing $\boldsymbol{v}, \boldsymbol{\beta}$ via the new variables $\boldsymbol{v}', \boldsymbol{\beta}'$, where each component v_i', β_i', is given as $v_i' = \frac{v_i}{v_i^k}$ and $\beta_i' = \frac{\beta_i}{v_i^k}$. This leads to the final program

$$\begin{aligned} \boldsymbol{\beta}', \boldsymbol{v}' &= \arg\min_{\boldsymbol{\beta},\boldsymbol{v}} \sum_{i=1}^{N} v_i' && (23) \\ \text{subject to} & \\ & \left\| \sum_{i=1}^{N} \alpha_i \phi_k(x_i) - \sum_{i=1}^{N} \underbrace{\beta_i' v_i^k}_{\beta_i} \phi_k(x_i) \right\|^2 = 0 && (24) \\ \text{and} \quad & -\beta_i' \leq v_i' \leq \beta_i'. && (25) \end{aligned}$$

The solution to this optimization problem are the vectors $\boldsymbol{v}', \boldsymbol{\beta}'$ with

$$\boldsymbol{\beta}' = R^{-1} S\alpha \quad \text{and} \quad \boldsymbol{v}' = |\boldsymbol{\beta}'|,$$

where $R_{i,j} = v_i^k k(x_i, x_j)$ and $S_{i,j} = v_i^k v_i^k k(x_i, x_j)$. If we start for example in the first iteration with $\boldsymbol{v}_i^1 = [1, \ldots, 1]$ we obtain the update equation

$$v_i^{k+1} = v_i^k |\beta_i'|.$$

This procedure typically converges in 10-20 steps. However, note that so far the minimization in (14),(15) will give a lossless compression. If we want to have a trade-off between sparsity and model reconstruction error we can adopt the same approach as in the ℓ_1-minimization (see section 4.2), and by a combined optimization functional

$$\arg\min_{\boldsymbol{\beta}} \left|\left| \sum_{i=1}^{N} \alpha_i \phi_k(x_i) - \sum_{i=1}^{N} \beta_i \phi_k(x_i) \right|\right|^2 + \gamma \sum_{i=1}^{N} \ln(\epsilon + |\beta_i|) \tag{26}$$

Like in ℓ_1-minimization, one is required to set the parameter γ a priori. The computational complexity of this optimization is higher than ℓ_1-minimization because one is required to solve multiple iterations of compression, rather than just one. However, on each successive iteration, there are less variables as one can *prune* the variables which have already obtained $\beta_i = 0$.

To compress multiple hyperplanes $\{w^1, \ldots, w^p\}$ we can easily generalize the previous algorithm. We simply learn $\beta^{(j)}$ for each hyperplane j, where we wish to minimize $||(\sum_j |\beta^{(j)}|)||_0$. To do this one has to repeat step (3) in algorithm A4.2 for each of the hyperplanes and calculate the rescaling weight by summing over all coefficient vectors, i.e.

$$v_i^{k+1} = (\sum_i |\beta_i^{(j)}|) v_i^k.$$

We show the algorithm for the exact compression in A4.3, keeping in mind that the extension to lossy compression is straightforward. Furthermore, note that this form of multi-output compression is computationally less demanding than the ℓ_1-minimization via equation (4).

4.3.4 Chunking Method: handling large datasets

All techniques so far require to operate on all training points at least once. For example in matching pursuit we need to search among all points for the next dictionary element.

Algorithm A4.2 ℓ_0-MINIMIZATION

Given a kernel matrix K, $w = \sum_i \alpha_i \phi_k(x_i)$ and training points $D_N = \{X, Y\}_{i=1}^N$

1. Set $v_i^1 = 1$,$k = 1$
2. Repeat until convergence
2. Obtain new solution β to the reconstruction equation in (23) by $\boldsymbol{\beta} = R^{-1} S\boldsymbol{\alpha}$.
3. Rescale Coefficients: $v_i^{k+1} = |\beta_i| v_i^k$.
4. k=k+1

Algorithm A4.3 MULTI-OUTPUT ℓ_0-MINIMIZATION

Given a kernel matrix K, p hyperplanes $w^p = \sum_i \alpha_i^p \phi_k(x)_i$ and training points $D_N = \{X, Y\}_{i=1}^N$

1. Set $v_i^1 = 1$,$k = 1$
2. Repeat until convergence
3. Obtain new solution $\beta^{(j)}$ to the reconstruction equation in (23) by $\boldsymbol{\beta}^{(j)} = {R^{(j)}}^{-1} S^{(j)} \boldsymbol{\alpha}^{(j)}$ for each hyperplane $j = 1, \ldots, p$.
3. Rescale Coefficients by the summed coefficients: $v_i^{k+1} = (\sum_i |\beta_i^{(j)}|) v_i^k$.
4. $k = k + 1$.

Since this can be computationally expensive, we propose a cascaded approach where we apply the proposed techniques to subsets and fuse the resulting reduced sets and possibly repeat. The linear expansion of the solution can be grouped without changing the result, namely:

$$\begin{aligned} w &= \sum_{i=1}^{N} \alpha_i \phi(x_i) \\ &= \sum_{i=1}^{N_1} \alpha_i \phi(x_i) + \sum_{i=N_1+1}^{N_2} \alpha_i \phi(x_i) \cdots \sum_{i=N_{n-1}+1}^{N} \alpha_i \phi(x_i), \\ &= \sum_{j=1}^{n} w^j, \end{aligned}$$

with $w^j = \sum_{i=N_{j-1}}^{N_j} \alpha_i \phi_k(x)_i$. The idea is to apply any of the previously described techniques (ℓ_1, ℓ_0 or Matching Pursuit) to a smaller problem, keeping only a subset of the variables active that we wish to compress, and leaving the rest fixed. We then move the active set to another subset, and iterate through the subsets. This means that we never face an optimization problem larger than a fixed size, of our choice. (However, if the chunk size is set too small compression will not be easy to perform as linearly or nearly linearly independent points may not exist in the active set.) Now, we can apply the introduced techniques to the smaller expansions w^j and combine their results. Furthermore, we can apply backfitting again to tune the final coefficients. Before we investigate all the introduced techniques let us comment on some basic pitfalls common to all approaches so far.

4.3.5 A common pitfall in reduced set selection

In general, a common feature of reduced set methods is the identification of construction of a small set of examples $Z = \{z_i\}_{i=1}^s \subset D_N$ and the re-parametrization of $w = \sum_{i=1}^N \alpha_i x_i$ in terms of Z by $\sum_i^s \beta_i z_i$. The re-parametrization can be done analytically by the solution of (2), namely

$$\beta = K_{Z,Z}^{-1} K_{Z,X} \alpha.$$

However, note that for most kernels, the reduced sets obtained from an ℓ_0 or ℓ_1 problem still might be almost linear dependent due to finite accuracy effects. For this reason $K_{Z,Z}$

might be badly conditioned, or even not invertible. The common approach to invert a regularized version of $K_{Z,Z}$ will lead to the problem that the regularization parameter depends on $K_{Z,Z}$ and this leads to an extra expensive model selection step. Furthermore, the standard pseudo-inverse leads to numerically unstable solutions for the expansion coefficients β. For this reason we propose to choose β by using a conjugate gradient based technique to minimize $||K_{Z,Z}\beta - K_{Z,X}\alpha||$ and to stop minimization by restricting the number of iterations. In practice this is as fast as calculating the pseudoinverse but due to ill-conditioning by far numerically much more stable.

4.4 Experiments

We conducted experiments on four types of data: (i) artificial data, (ii) two-class classification problems, (iii) multi-class classification problems and (iv) regression problems. The summary of the datasets we use, apart from the artificial one of Section 4.4.1, are given in Table 4.1. For each dataset we describe the machine learning algorithm that is used for learning, along with its parameters, in Table 4.2. These are the models on which we perform reduced set selection. The parameters selected for German, Image, Waveform and Banana come from [70]. For USPS and Letter we tried to choose parameters which matched the test error quoted elsewhere [28, 95]. For the regression datasets, Abalone and Kin-32nm, we tried to pick the best possible parameters on the test set.[4] Apart from the first toy dataset, in all cases we used RBF kernels. Although performing reduced set selection on kernels that are not full rank such as polynomial kernels can result in compression with exactly the same decision rule, we decided to stick to the more difficult case of approximating a kernel with full rank.

In our experiments with hyperplane matching pursuit, we perform backfitting, and optimize the criterion $||w - \hat{w}||^2$. For all methods we also optimize the threshold b for the classification tasks by explicitly minimizing the training error.

4.4.1 Artificial Problems

We first constructed artificial data by generating two classes from two Gaussian clouds in 10 dimensions with means $(1, 1, 1, 1, 1, 0, 0, 0, 0, 0)$ and $(-1, -1, -1, -1, -1, 0, 0, 0, 0, 0)$ and standard deviation 4 following [2]. We trained a linear SVM for differing amounts of

[4] RSS requires that we have given an estimator first. To this end we tried to pick the parameters as optimal as possible on the test set.

Name	Inputs	Outputs	Train	Test
German	20	1	700	300
Waveform	21	1	1000	4000
Banana	2	1	1000	4300
Image	18	1	2000	300
USPS	256	10	7329	2000
Letter	17	26	16,000	4000
Abalone	8	1	3133	1044
Kin-32nm	32	1	3000	5192

Table 4.1: Datasets used in the experiments. All the datasets are classification problems, apart from Abalone and Kin-32nm which are treated as regression problems.

Name	Algorithm	SVs	Test Err
German	SVM ($\sigma = 5.24, C = 3.16$)	400	0.2570
Waveform	SVM ($\sigma = 3.16, C = 1$)	331	0.0917
Banana	SVM ($\sigma = 0.7, C = 316$)	220	0.1016
Image	SVM ($\sigma = 3.9, C = 500$)	216	0.0281
USPS	SVM ($\sigma = 128, C = 1000$)	1526	0.0416
Letter	SVM ($\sigma = 4, C = 1000$)	4522	0.0373
Abalone	RR ($\sigma = 15, \gamma = 1e-5$)	3133	0.410
Kin-32nm	RR ($\sigma = 20, \gamma = 0.005$)	3000	0.6187

Table 4.2: Predictors used in the experiments. Included are the parameters of the initial trained predictors we are trying to compress. The error rates for the first four datasets are averaged over ten splits. The last two datasets, Abalone and Kin-32nm, have error rate measured using mean squared error (after normalizing the outputs).

training points. This is an unrealistic case, as one can represent the data in primal form by calculating w in this case, nevertheless it serves to show that SVMs do not optimally compress in dual space. In this case, any of the reduced set selection methods (with appropriate hyper-parameter choice) will choose 10 SVs with no loss in accuracy. The results are given in Figure 4.1 using the ℓ_0-minimization method with $\gamma = 0$. Note the linear increase in number of SVs for SVMs in comparison to the fixed number of SVs which is independent of training set size for reduced set methods. This is due to the fact that all mislabeled points become SVs in the SVM, and for large datasets this has a significant effect on computation time. Indeed the fraction of SVs in the SVM is lower bounded by the number of training errors, and hence asymptotically by the Bayes error, as pointed out in [87].

4.4.2 Two-class Classification

We took four different datasets and trained SVMs with the parameter choices quoted in [70]. We then compared the error rate to compression ratio of the competing methods: ℓ_1- minimization, ℓ_0- minimization and hyperplane matching pursuit. For ℓ_1- and ℓ_0- minimization we chose the best value of γ on the data set from an array of parameter choices for the trade-off parameter γ. The resulting number of SVs we used as a maximum iteration constraint for hyperplane matching pursuit. All experiments are performed in a cross-validation setting where we used ten splits. The resulting number of SVs is averaged over the ten splits since the parameter γ did not result in the same number of SVs each time. The results, shown in Table 4.2 indicate similar performance between the methods, although hyperplane matching pursuit yields slightly lower test error for higher compression rates.

The time needed for computing the ten parameter choices for each of the methods is given in Table 4.3. We show this for ten parameter choices because to find a good loss to compression ratio a hyper-parameter search is necessary. For example, one can evaluate accuracy using a validation set. As hyperplane matching pursuit essentially gives all parameter choices at no extra cost through a single run of greedy minimization, it yields much faster execution times. Moreover, the hyper-parameter γ in the other two methods is difficult to control - it is difficult to know which value yields which number of SVs. We also analyzed the training error and value of the objective function for all three algorithms. This can be seen for the Waveform dataset in Table 4.3. We found

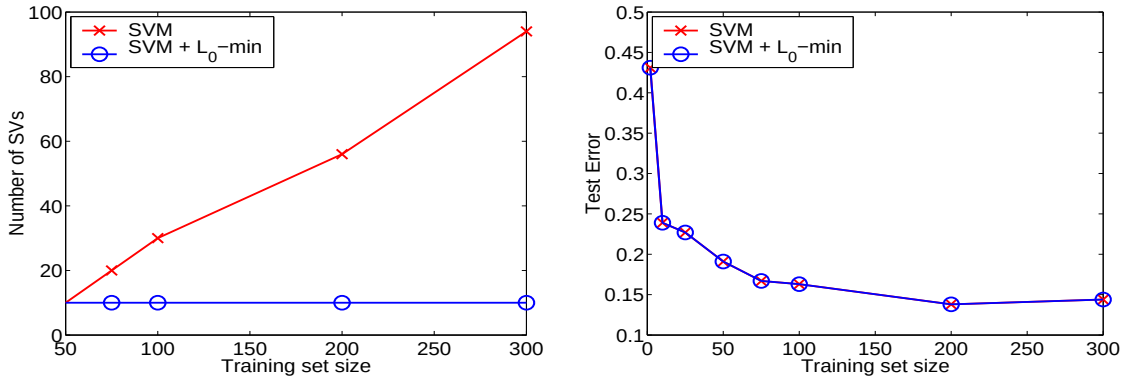


Figure 4.1: Reduced Set Selection on Artificial data. The number of SVs increases linearly for standard SVM due to *margin errors* becoming SVs, but is constant for RSS methods such as ℓ_0-minimization.

that the objective function and training error were lower for hyperplane matching pursuit particularly in the case of high compression rates, mirroring its test error performance.

Note that it is easier to compress the expansions on the problems with higher error rates (Image, which has the smallest test error, is the hardest to compress.) This corroborates the results found with artificial data in Section 4.4.1.

4.4.3 Multi-Class Classification

We then performed reduced set selection on multi-class problems. In this setting, one might expect to achieve a greater gain if the reduced set selection is coupled across the different classifiers. We tested the compression of SVM solutions using the one-against-the-rest method [22] on two datasets: USPS Digit Recognition and the Letter database, with 10 and 26 classes respectively. We performed ℓ_1-minimization on SVM solutions for a range of different values of γ (the compression parameter.) The results are given in Figure 4.4. We performed Matching Pursuit using the same number of SVs for each hyperplane as the ℓ_1 method to enable direct comparison. Again, although SVMs produce sparse solutions, these can be compressed considerably by methods such as ℓ_1-minimization or the Matching Pursuit method. The latter again outperforms the former for higher compression rates. We show results for the "coupled" Multi-hyperplane Matching Pursuit method which finds SVs which are relevant to all hyperplanes (for each class) at once. The compression rate is significantly improved. We did not compare with the algorithm given in (3) because it was too slow to compute, however in [79] the authors do report a single run of this system with an error rate of 5.5% for 570 SVs which they compare to 10.8% using the ℓ_1-minimization approach of equation (4) which does not take into account the multi-class problem.

Dataset	German	Image	Waveform	Banana
ℓ_1-min	613 secs	152 secs	290 secs	136 secs
ℓ_0-min	645 secs	260 secs	250 secs	131 secs
M-Pursuit	21 secs	16 secs	10 secs	2 secs

Table 4.3: Calculation time of reduced set selection methods for two-class classification for ten parameter choices.

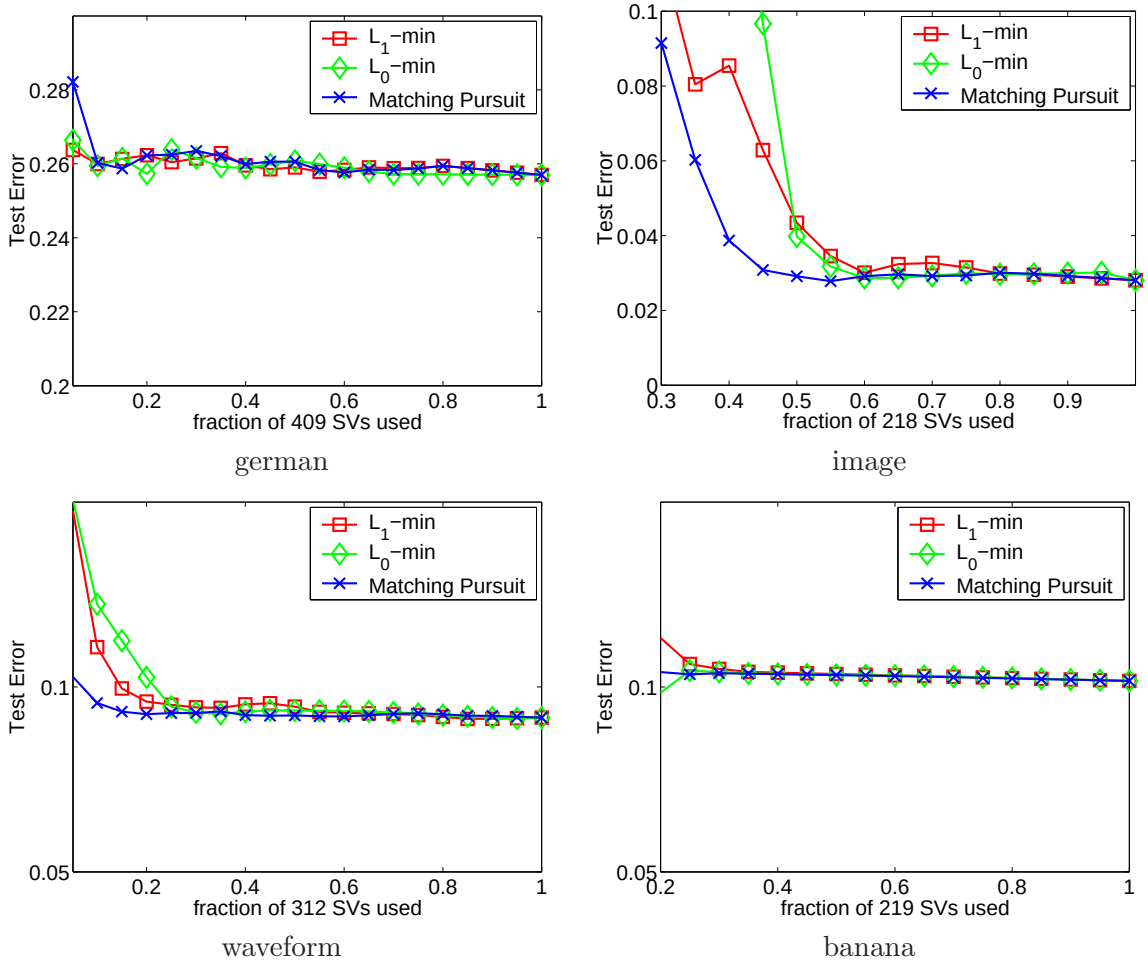


Figure 4.2: Comparison of Reduced Set Selection Methods for Two-Class Classification. The three reduced set selection methods perform similarly, but the Hyperplane Matching Pursuit method yields lower test error for high compression rates on Waveform and Image.

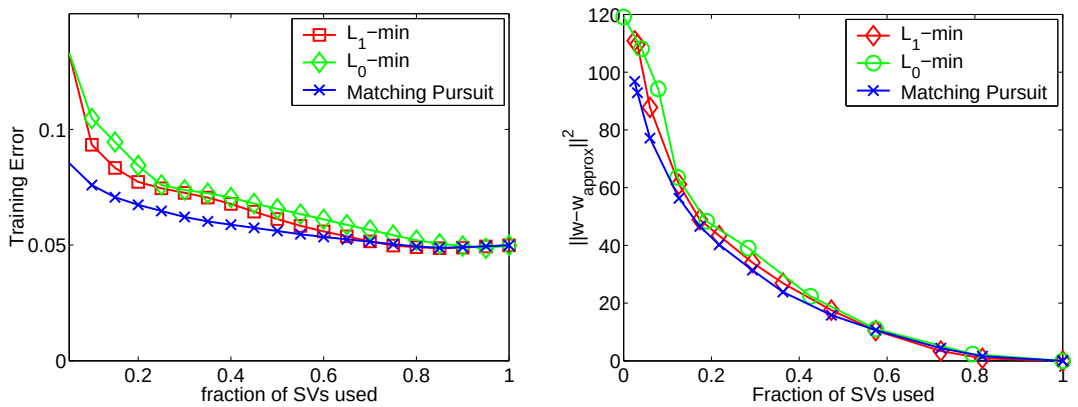


Figure 4.3: Training error and objective function value on the waveform dataset. The training error (left) and value of the objective function $||w - \hat{w}||^2$ (right) are both lower for the Hyperplane Matching Pursuit method relative to the other two methods.

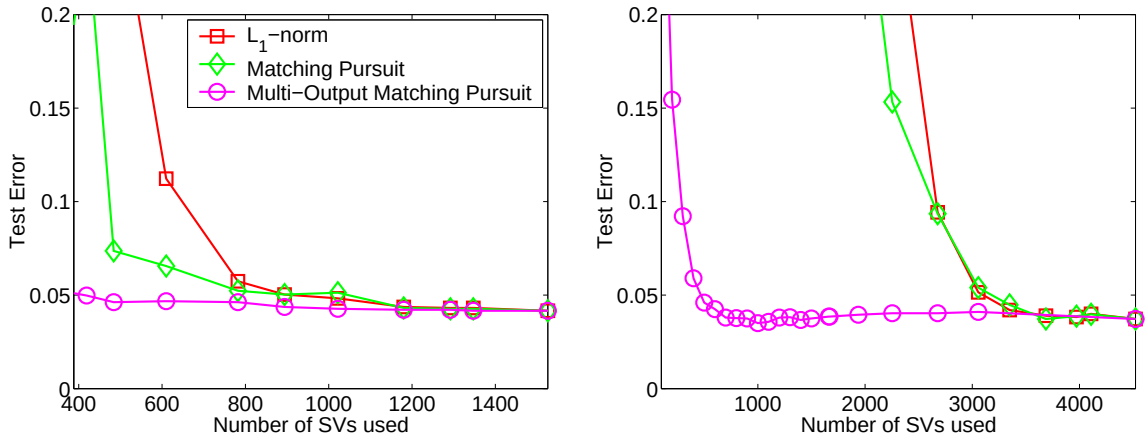
We can expect these results to carry over to similar settings such as regression with multiple outputs. Moreover, we can expect larger gains on other multi-class methods such as one-vs-one or error correcting codes [22] which use more support vectors.

4.4.4 Regression

We performed experiments on the two datasets mentioned before in a regression setting. We attempted to compress the solution of Ridge Regression in dual variables which does not give sparse solutions, hence the potential gain here should be much greater than in SVM. Our experiments indicate that this was indeed the case. We normalized both input and outputs to have mean zero and standard deviation one, and give the mean squared error on the normalized outputs. Figure 4.5 shows the error rate for differing compression rates using Hyperplane Matching Pursuit only. The results show that compressing RR solutions can give significant efficiency gains, and we expect this result to hold for other non-sparse predictors as well, such as the Nadaraya-Watson Estimator or Parzen's Windows [22], to name two.

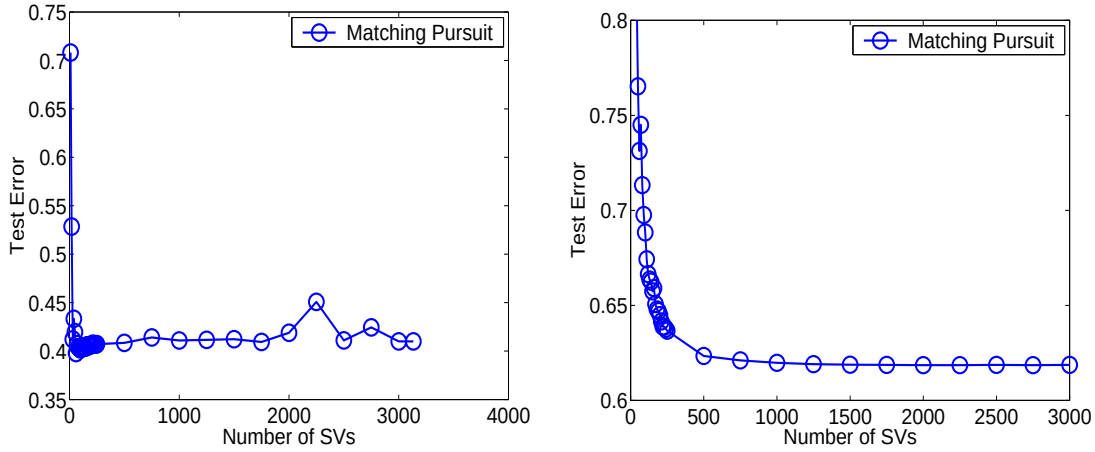
4.4.5 Reduced Set Selection for Kernel PCA

As mentioned before, it is likely that kPCA is well suited for the application of reduced set selection since the solution is expressed in terms of all data points. In this section, we demonstrate that Multi-Output Matching Pursuit is well suited for reduced set selection for kPCA. This is critical for KDE since its extensive usage of kPCA. In fact we will use Multi-Output Matching Pursuit later in chapter 6 to compress kPCA to tune KDE for a real-time robotic application. Here, we present results on a synthetic toy data set. Our goal is to give qualitative evidence that Multi-Output Matching Pursuit can be used for kPCA compression. Let us generate data by sampling from three Gaussian clusters located at $c^1 = [-1, -1], c^2 = [2, 1], c^3 = [2, -2]$. We use a Gaussian kernel with $\sigma = 0.6$ and apply kPCA to obtain two principal coordinates r_1^i, r_2^i of sample x_i in feature space. Since we can not directly look at the principal directions e^1, e^2, we will visualize them indirectly. This is done by considering a dense grid $[-4, 4]^2$ in the input space. Let g_i denote such a grid point. We project each g_i on the principal directions via evaluating $e_1^\top \phi_k(g_i)$ and $e_2^\top \phi_k(g_i)$. Note that these projections and generally the principal coordinates of input points must be always smaller than one when a Gaussian kernel is used. We can interpret each principal coordinate of a grid point as an intensity and plot the grid as an image. A higher intensity of point g_i implies that the feature point $\phi(g_i)$ is more collinear to the principal direction. In figure 4.6 we show the obtained intensity maps where the left column is obtained by projection on the first principal direction and the second column is obtained by projection on the second principal direction. In the first row we show the original solution obtained by kPCA where we plotted each input point which is used in the expansion for kPCA. In the next row we show the projection obtained by using the subset of points after applying Multi-Output Matching Pursuit. Note that even after keeping only 5 % of the original data points we are able to obtain almost the same principal directions. In figure 4.7 we show the normalized reconstruction error obtained for the penalties $\gamma \in \{2^{-10}, 2^{-9.5}, 2^{-9}, \ldots, 2^0\}$ used in Multi-Output Matching Pursuit where we plot the number of retained points vs the reconstruction error. As reconstruction error we use the squared loss of the principal coordinates obtained by using the original principal direction e^i and the new approximated direction $\hat{e}^i$. For better visualization we normalized the result by the maximum error (which corresponds to the sparsest expansion).



postal letter

Figure 4.4: Comparison of Reduced Set Selection for Multi-Class Classification. The Hyperplane Matching Pursuit Algorithm designed for Multi-Class problems reduced the number of kernel computations required on the USPS Postal Database (left) and UCI Letter Database (right) compared to binary classification-based compression. One can see that we obtain high compression rates and a low test error.



Abalone Kin-32nm

Figure 4.5: Matching Pursuit for Compression of Ridge Regression. The Hyperplane Matching Pursuit Algorithm reduced the number of kernel computations required on the Abalone (left) and Kin-32nm (right) problems compared to standard Ridge Regression, which is not sparse.

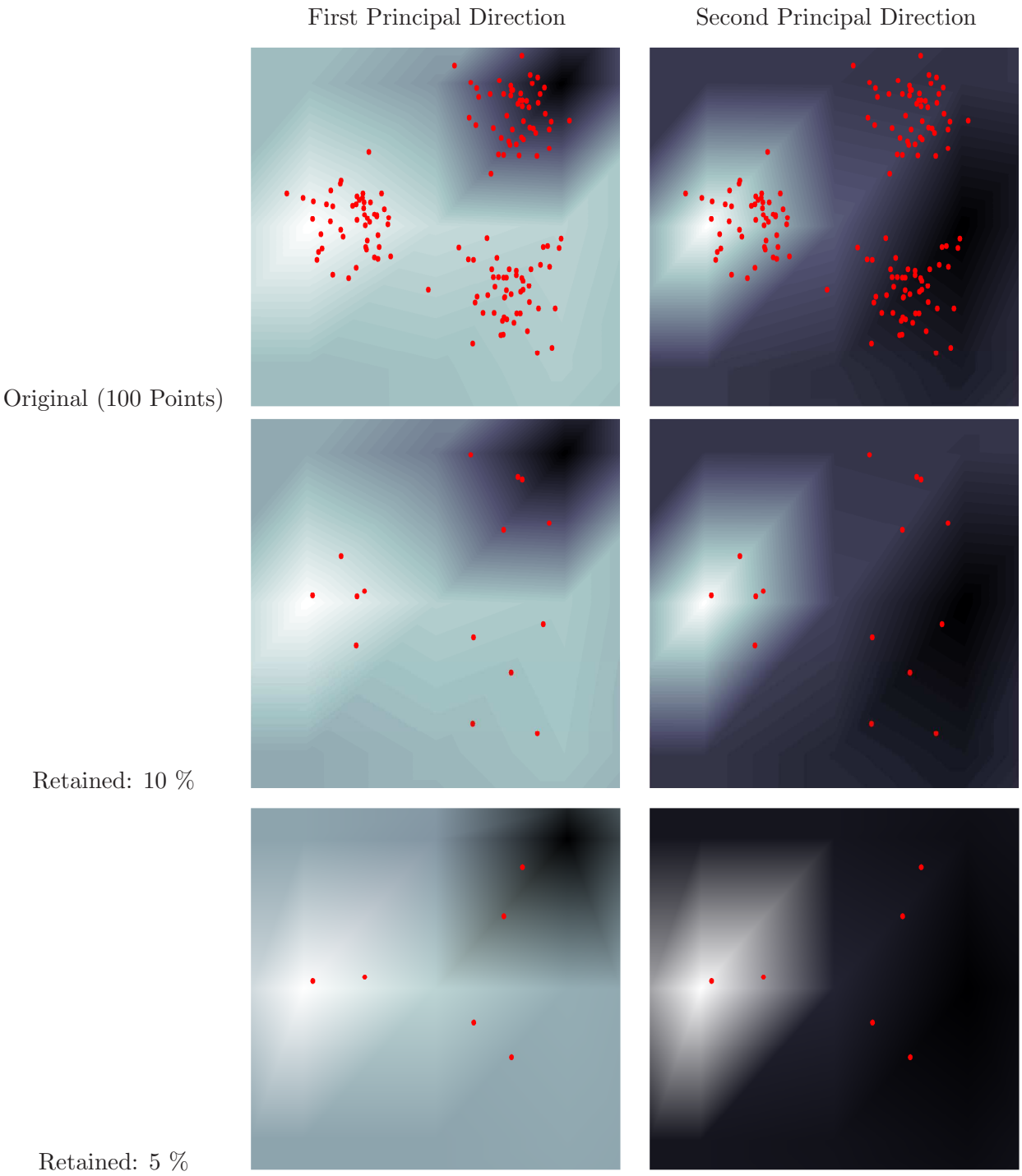


Figure 4.6: Compression of Kernel PCA. One can see that by only 5% of the original data points, qualitatively the same feature extractors are obtained.

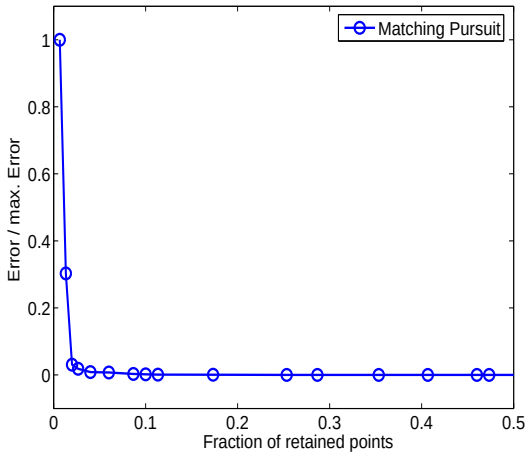


Figure 4.7: Compression of Kernel PCA obtained by varying the penalty term γ used in Multi-Output Matching Pursuit. We plot the number of kept expansion points vs reconstruction error.

4.5 Conclusion

Reduced set techniques fall into two categories: reduced set selection and reduced set construction. Reduced set construction tries to reduce the amount of kernel points by constructing a new set of points which are not necessarily a subset of the training data. In contrast, reduced set selection methods try to reduce the amount of points by *selecting* a linear independent subset of the training data. Often the compression factor in reduced set selection is worse than in reduced set construction but in general reduced set selection methods are more efficient and numerically more stable than reduced set construction techniques. Another reason why we prefer reduced set selection is that it is readily available in a typical KDE scenario where the input or output domains are possibly not differentiable. In such a case one can only use the current dataset for compression purposes whereas *construction* of new candidates is difficult.

We have reviewed and proposed optimization strategies for reduced set selection which can be solved efficiently and are also applicable for large scale problems. Our initial goal was to find more efficient techniques of achieving compression, and while both Hyperplane Matching Pursuit and ℓ_0-minimization present viable novel alternatives we prefer the former due to

(1) reduced computational complexity,

(2) the ability to compute all smaller expansions up to the described one, and

(3) its good performance at high compression rates.

In the multi-output case, significant gains can be made in terms of compression by coupling the compression of all trained hyperplanes at once. This led to high compression levels in the USPS and Letter datasets and in kPCA which is crucial for time-critical usage of KDE. Furthermore, this result suggests that one could also gain significant speedups in the training phase by simultaneously training several classificators. The most straightforward way of achieving such a goal would be to adapt the kernel matching pursuit algorithm for direct multi-class classification. This is a topic of future research.

Part II

Applications to robotics

Chapter 5

The problem of robot imitation – Optimization based Approach

Imitation learning is a promising technique for teaching robots complex movement sequences. One key problem in this area is the transfer of perceived movement characteristics from perception to action. For the solution of this problem representations are required that are suitable for the analysis and the synthesis of complex action sequences. In this chapter we describe the method of Hierarchical Spatio-temporal Morphable Models for a representation of complex movement sequences. This method allows an automatic segmentation of movements sequences into movement primitives, and a modeling of these primitives by morphing between a set of prototypical trajectories. We apply this method to imitation learning and to the synthesis of human writing movements. The synthesized movements are subsequently transferred to a human-like robot arm.

5.1 Introduction

The goal of imitation learning is to teach robots movement sequences by observation of a teacher. Imitation learning has to address two fundamental problems.

1 The movement characteristics of observed movements have to be transferred from the perceptual level to the level of generated actions (see [77] and [59]).

2 Continuous spaces of movements with variable *styles*, i.e. different movement performances belonging to the same class, have to be approximated based on a limited number of learned example sequences.

   This implies that the robot should be able to synthesize new movements based on the learned examples.

One method that fulfills these requirements is the technique of Spatio-Temporal Morphable Models (STMMs). This methods represents the spatio-temporal characteristics of complex movement sequences by linear combinations of example trajectories with different characteristics. Linear combinations of space-time patterns can be defined efficiently by exploiting spatio-temporal correspondence, by weighted summation of spatial and temporal displacement fields that morph the prototypical movement trajectories into a reference pattern. This method has been successfully applied by [11] and [32] for the generation and analysis of complex movements in computer graphics as well as for the recognition of movements and movement styles from trajectories in computer vision by [32, 40].

To generalize the method of linear combination for complex sequences containing multiple complex movements we have extended the basic STMM algorithm by introducing a second hierarchy level that represents motion primitives. Each movement primitive is modeled using a STMM. In this way generative models for complex sequences of movements with variable styles can be learned from example trajectories. This method of hierarchical STMMs (HSTMMs) has been successfully applied for the automatic recognition and synthesis of sequences of complex karate techniques by [40], and for the estimation of skill levels of different actors by [41] using a small amount of motion capture data. This shows that STMM is suitable for building models for continuous movement spaces from small amount of training data that can be used for analysis and synthesis.

In this chapter we present an application of this algorithm for the Imitation Learning in robots. We show how HSTMMs can be linked with a robot control architecture. We illustrate our method by imitating human-like writing movements using a robot arm. Based on a small number of prototypical examples our robot can learn to imitate and caricature writing styles, and to synthesize similar styles of writing movements.

5.1.1 Related Work

Our work includes the identification and the segmentation of movement primitives, and the low-dimensional representation of movements by interpolation. Multiple methods for the parameterization of movement styles have been proposed in computer graphics and computer vision, e.g. based on Hidden Markov Models by [7] and [100], principal component analysis by [102],[5] and [10], or Fourier components as in [94]. Different studies on imitation learning have investigated methods for describing the spatio-temporal characteristics of movements using Principal Component Analysis by [25] and Spatio-temporal Isomaps by [43]. In [71] a verb-adverb approach was proposed that applies a combination of radial basis functions and low-order polynomials for defining parameterized interpolations between example movements. For this approach specific key times (e.g. the foot contact with the ground) must be specified by hand. Time Warping is defined by linear interpolation between these key times. In [93] and [63] this interpolation is realized with splines.

For the identification and segmentation of movement primitives within longer movement sequences appropriate features are required, that provide a robust characterization of individual movement elements. Different elementary spatio-temporal and kinematic features have been proposed in the literature, like angle velocity as was proposed in [65], or curvature and torsion of the 3D trajectories as in [14].

5.2 Hierarchical Spatio-temporal Morphable Models as representation for Imitation Learning

An overview of the algorithm is shown in figure 5.1. The following sections describe the extraction of the movement elements, the modeling by STMMs, and the transfer of synthesized movement sequences onto the robot arm.

For the identification of movement primitives within a complex movement sequence an appropriate description of the spatio-temporal characteristics of the individual movement elements must be found that are suitable for a robust matching with stored example templates. We utilize an algorithm which uses dynamic programming to find the movement primitives. A formal description of the algorithm is given in [40].

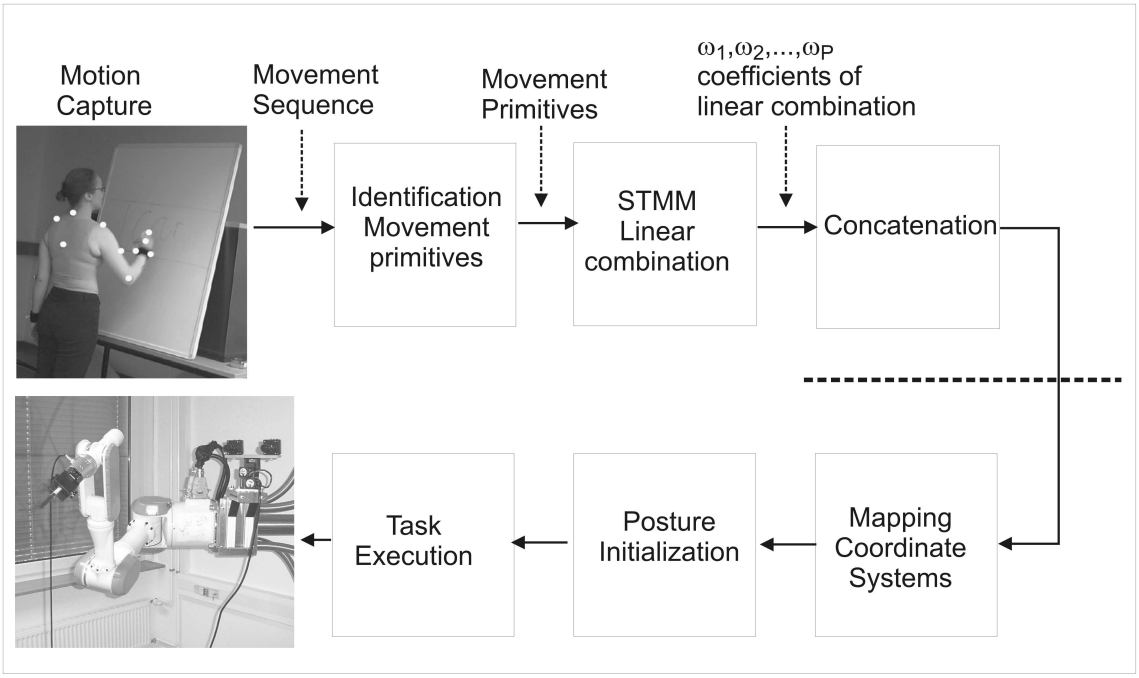


Figure 5.1: Schematic description of the algorithm to synthesize and imitate complex movement sequences. In the first step the sequence is decomposed into movement primitives. These movement primitives can be analyzed and changed in style by defining linear combinations of prototypes with different linear weight combinations. Afterward the individual movement primitives are concatenated into longer movement sequences. This technique allows to generate sequences containing movements with multiple styles. The mapping of these movement sequences onto the robot arm is done in three steps: mapping of coordinates, posture initialization, and task execution.

5.2.1 Morphable Models for modeling movement primitives

The technique of *spatio-temporal morphable models* (see [31, 32]) is based on linearly combining the movement trajectories of prototypical motion patterns in space-time. Linear combinations of movement patterns are defined on the basis of spatio-temporal correspondences that are computed by dynamic programming. Complex movement patterns can be characterized by trajectories of feature points. The trajectories of the prototypical movement pattern p can be characterized by the time-dependent vector $\boldsymbol{\zeta}_p(t)$. The correspondence field between two trajectories $\boldsymbol{\zeta}_1$ and $\boldsymbol{\zeta}_2$ is defined by the spatial shifts $\boldsymbol{\xi}(t)$ and the temporal shifts $\tau(t)$ that transform the first trajectory into the second. The transformation is specified mathematically by the equation:

$$\boldsymbol{\zeta}_2(t) = \boldsymbol{\zeta}_1(t + \tau(t)) + \boldsymbol{\xi}(t) \tag{1}$$

By linear combination of spatial and temporal shifts the spatio-temporal morphable model allows to interpolate smoothly between motion patterns with significantly different spatial structure, but also between patterns that differ with respect to their timing.

The correspondence shifts $\boldsymbol{\xi}(t)$ and $\tau(t)$ are calculated by solving an optimization problem that minimizes the spatial and temporal shifts under the constraint that the temporal shifts define a new time variable that is always monotonically increasing (fig. 5.2). For further details about the underlying algorithm we refer to [31] and [32].

Signifying the spatial and temporal shifts between prototype p and the reference pattern by $\boldsymbol{\xi}_p(t)$ and $\tau_p(t)$, linearly combined spatial and temporal shifts can be defined by the two equations:

$$\boldsymbol{\xi}(t) = \sum_{p=1}^{P} w_p \, \boldsymbol{\xi}_p(t) \qquad \tau(t) = \sum_{p=1}^{P} w_p \, \tau_p(t) \tag{2}$$

The weights w_p define the contributions of the individual prototypes to the linear combination. We always assume convex combinations with $0 \leq w_p \leq 1$ and $\sum_p w_p = 1$. After linearly combining the spatial and temporal shifts the trajectories of the morphed pattern can be recovered by morphing the reference pattern in space time using the spatial and temporal shifts $\boldsymbol{\xi}(t)$ and $\tau(t)$. The space-time morph is defined by equation (1) where $\boldsymbol{\zeta}_1$ is the reference pattern and $\boldsymbol{\zeta}_2$ has to be identified with trajectory of the linearly combined pattern.

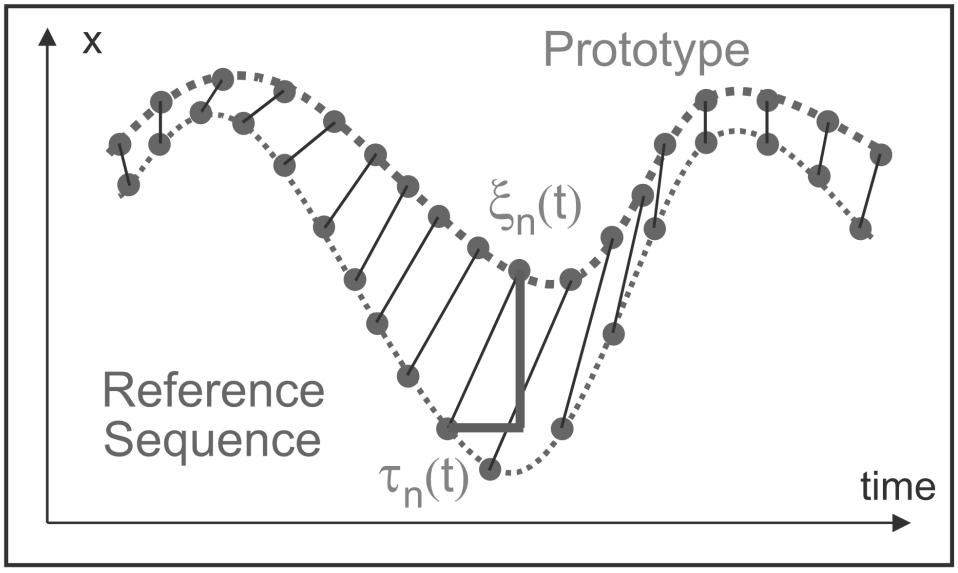


Figure 5.2: Illustration of the established spatio-temporal correspondence between a prototypical trajectory and a test sequence with the correspondence vector fields τ and ξ.

5.2.2 Concatenation

The synthesized movement elements were concatenated using an algorithm described in [33]. The trajectory of each movement element is decomposed into a "normalized" trajectory and its start and endpoint. The normalized trajectory is zero at the transition points between the elements. Start points, end points, and the normalized trajectories are linearly combined separately.

5.3 Transferring human-like movements to a robot arm

The transfer of the trajectories to the robot is performed in three stages:

1) The HSTMM synthesizes trajectories in the same space as the prototype trajectories. Therefore, one has to transform synthesized trajectories from the prototype space into the task space of the robot. Also the trajectory is scaled appropriately.

2) The second stage initializes the robot posture to a specific recorded (and appropriately transformed) initial human arm posture.

3) The task execution is performed by reproducing the exact end-effector trajectory and approximating the human arm posture.

5.3.1 Mapping of the coordinate systems

In the investigated task of writing movements the end effect trajectories are approximately planar. The drawing area of the synthesized writing movements has to be transformed into a drawing area in task space. The drawing plane is given[1] by two vectors $\mathbf{u}$ and $\mathbf{v}$, which define a task orientation frame

$$\mathbf{T}_t = [\ \mathbf{u} \quad \mathbf{v} \quad \mathbf{u} \times \mathbf{v}\ ]. \tag{3}$$

The starting point of the movement is given by the position vector $\mathbf{p}$. . Since the task space is planar, we can use the first principal components $\mathbf{e}_1, \mathbf{e}_2$ of the HSTMM output sequence $\zeta(t)$, to define an orientation frame of the trajectory as

$$\mathbf{T}_d = [\ \mathbf{e}_1 \quad \mathbf{e}_2 \quad \mathbf{e}_1 \times \mathbf{e}_2\ ]. \tag{4}$$

Note that $\mathbf{e}_1, \mathbf{e}_2$ span the whole task space for our application. The trajectory $\zeta(t)$ is then first centered

$$\hat{\zeta}(t_i) = \zeta(t_i) - \frac{1}{N}\sum_{k=1}^{N} \zeta(t_k), \tag{5}$$

where we assume that the trajectory is given in a discretized form $\zeta(t_1), \ldots, \zeta(t_N)$ with $t_1 = 0$. The centered trajectory $\hat{\zeta}(t_i)$ can be scaled to avoid violation of task space constraints. The final target trajectory $\boldsymbol{\zeta}^\star(t)$ is given by

$$\boldsymbol{\zeta}^\star(t) = \mathbf{p} + \mathbf{T}_t \mathbf{T}_d^{-1} \left(\hat{\zeta}(t) - \hat{\zeta}(0) \right). \tag{6}$$

5.3.2 Initialization of robot posture

The kinematic structure of humans and robots are usually different. Therefore, marker positions can usually not be transferred to the robot directly. Only if the robot is humanoid and has an equivalent kinematic structure the marker positions can be transferred directly, see [69]. Otherwise one has to define "posture specifiers" that are applicable to humans as well as to robots. Imitation of posture is achieved by transferring these posture specifiers from the human to the robot.

Let $\mathbf{L}_d, \mathbf{R}_d, \mathbf{E}_d$ and $\mathbf{F}_d$ denote left shoulder, right shoulder, elbow and finger marker in transformed prototype space. As posture specifiers we chose orientation normals of two planes. The normal vector of the first plane is defined as

$$\mathbf{e}_d = \frac{(\mathbf{L}_d - \zeta^\star(t_1)) \times (\mathbf{E}_d - \zeta^\star(t_1))}{\| (\mathbf{L}_d - \zeta^\star(t_1)) \times (\mathbf{E}_d - \zeta^\star(t_1)) \|}. \tag{7}$$

This plane is spanned by the left shoulder, the elbow and an arbitrary reference point. In our case we chose the starting point $\zeta^\star(t_1)$ of the trajectory $\zeta^\star(t)$. Equivalently let

$$\mathbf{f}_d = \frac{(\mathbf{R}_d - \zeta^\star(t_1)) \times (\mathbf{F}_d - \zeta^\star(t_1))}{\| (\mathbf{R}_d - \zeta^\star(t_1)) \times (\mathbf{F}_d - \zeta^\star(t_1)) \|} \tag{8}$$

[1] The plane could be determined by a stereo vision system.

be the normal of the second plane which is spanned by finger, right shoulder and $\zeta^\star(t_1)$. Let $\mathbf{q} = [\mathbf{q}_1, \mathbf{q}_2]$ be the joint values of the robot, where $\mathbf{q}_1$ influence the elbow position and $\mathbf{q}_2$ does not. The corresponding plane normals $\mathbf{e}_r(\mathbf{q}_1)$, $\mathbf{f}_r(\mathbf{q}_2)$ of the robot are calculated in an equivalent way (see fig. 5.3). For this purpose we use the a-priori specified position vector $\mathbf{p}$ from 5.3.1 instead of $\zeta^\star(t_1)$[2]. In addition a left virtual shoulder position has to be specified to determine the relative orientation of robot arm to the robot basis.

The initial posture of the robot is adjusted to the initial human posture by first minimizing

$$\min_{\mathbf{q}_1} \ \|\mathbf{e}_d - \mathbf{e}_r(\mathbf{q}_1)\|. \tag{9}$$

over the joints $\mathbf{q}_1$ and subsequently minimizing

$$\min_{\mathbf{q}_2} \ \|\mathbf{f}_d - \mathbf{f}_r(\mathbf{q}_2)\| \tag{10}$$

over $\mathbf{q}_2$. We minimize therefore the angles between $\mathbf{e}_r, \mathbf{e}_d$ and $\mathbf{f}_r, \mathbf{f}_d$ respectively.

5.3.3 Task Execution

Starting from its initial posture, the trajectory of the robot is planned by solving the following optimization problem that depends on the discretely sampled joint variables $\mathbf{q}(t_i)$:

$$\min_{\mathbf{q}(t_i)} \rho(\mathbf{q}(t_i)) = \|\mathbf{e}_d - \mathbf{e}_r\|^2 + \alpha\|\mathbf{q}(t_i) - \mathbf{q}(t_{i-1})\|^2$$
$$\text{subject to } \mathbf{P}_r(\mathbf{q}(t_i)) - \zeta^\star(t_i) = 0$$

where $\mathbf{P}_r(\mathbf{q}(t_i))$ describes the end-effector position. This is done for each time step t_i of the trajectory separately. The objective function $\rho(\mathbf{q}(t_i))$ measures the Euclidean distance between the normals $\mathbf{e}_d$ and $\mathbf{e}_r$. An additional regularization term is added to penalize high joint velocities. This term depends on the difference between the new joint configuration $\mathbf{q}(t_i)$ and the previous configuration $\mathbf{q}(t_{i-1})$. The scalar α determines the trade-off between smoothness of obtained joint trajectories and the quality of imitation. As a starting point, we use the joint values obtained by classical inverse kinematics. The joint trajectories were computed off-line. [3]

5.4 Experiments

We demonstrate the application of the proposed method by imitation and synthesis of human writing movements. In the following we describe the technical details and results of the performed experiments.

Motion Capture: We recorded writing movements of two human actors who wrote the the word "ICAR" (fig. 5.4) using a commercial motion capture system (VICON 612, Oxford) with 6 cameras. We used 10 (passive) markers that included the shoulders, 2 front and one rear torso, upper arm, elbow, front arm, hand and index finger of the writing arm.

[2] The reference point $\zeta(t_1)$ has to ensure that $\mathbf{e}_d \neq f_d \forall t$. If it does not, another reference point has to be chosen.

[3] A computational faster implementation to solve (5.3.3) is obtained by using explicit information about the null space of the manipulator Jacobian (see [82]).

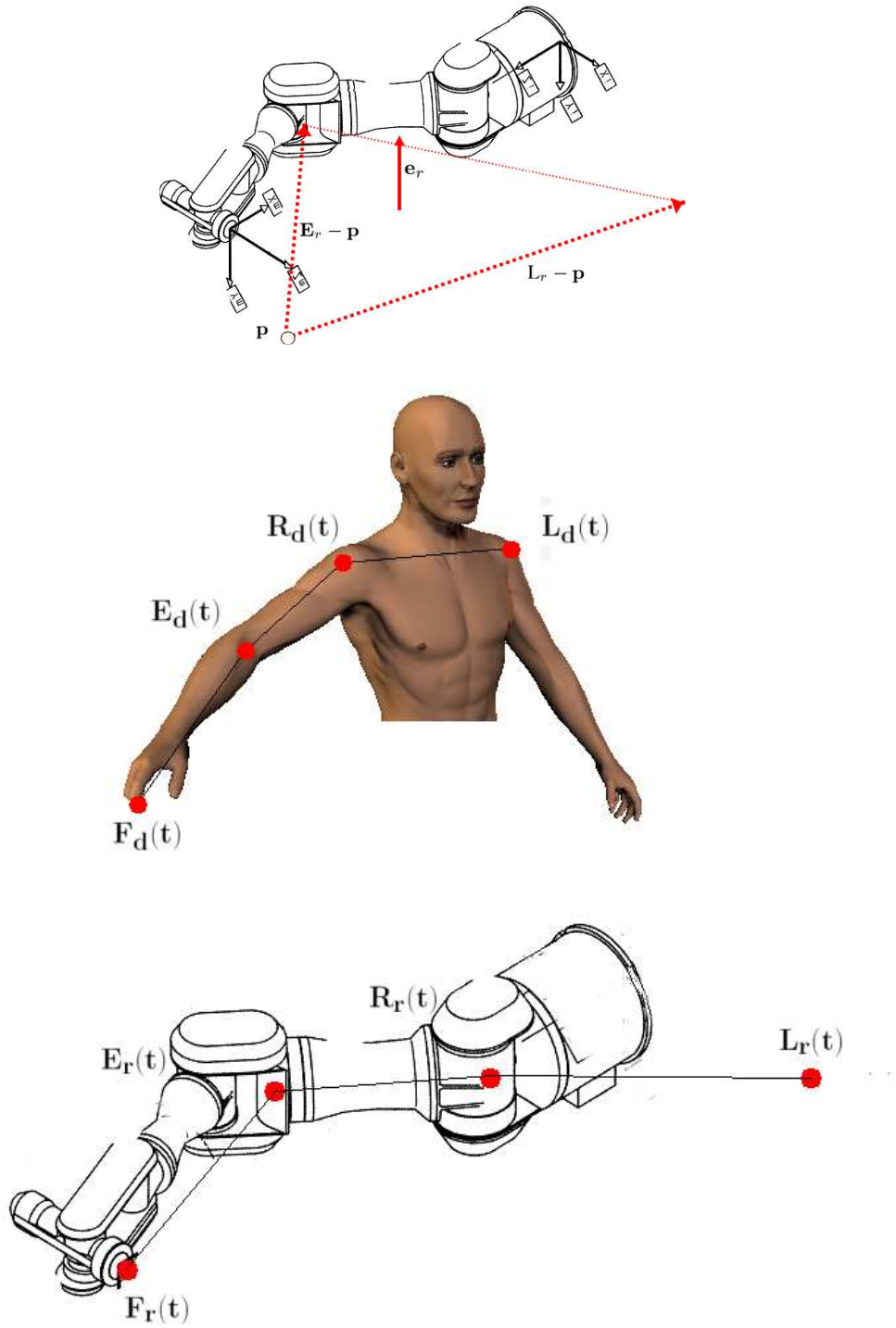


Figure 5.3: Illustration of the plane normals $\mathbf{e}_r$ and $\mathbf{f}_r$. A virtual left shoulder $\mathbf{L}_r$ position of the robot is defined a-priori.

Syntheses of writing movements: Continuous spaces of individual movements are generated by linear combinations of the segmented movement primitives. These movements are then automatically concatenated into longer sequences including multiple move-

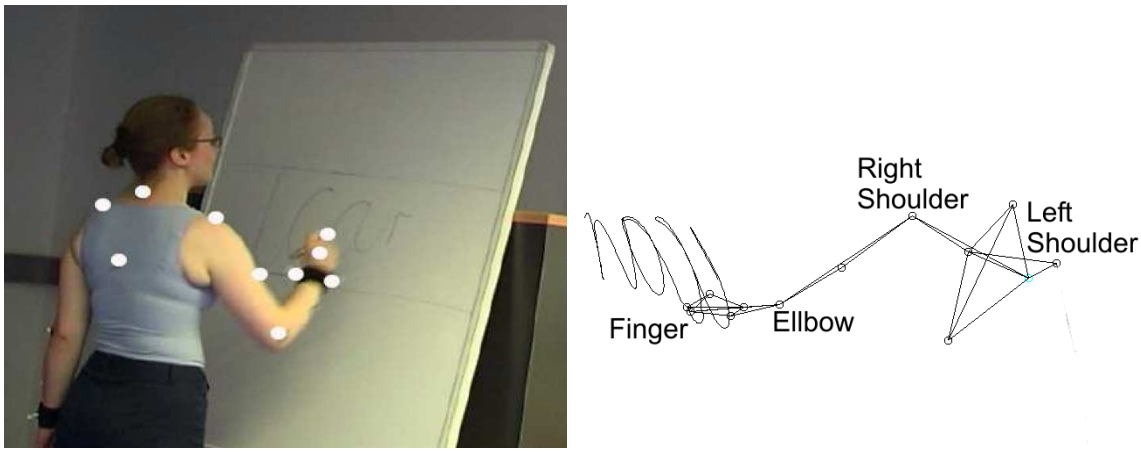


Figure 5.4: Left: Motion capturing of writing movements on a board. Right: Illustration of the marker set and the trace of the finger marker during the writing of the word "ICAR".

ment primitives. Figure 5.5 shows the synthesized pen trajectories of the writing movements. The method allows to morph continuously between the writing sequences of the two actors (left panel). In addition we can synthesize caricatures of the specific writing styles of each actor (right panel). Also, the individual movement primitives can be reassembled in a different sequential order, e.g. in order to write the word "'IACR" (middle row). All movement sequences were synthesized based on only two prototypical example trajectories.

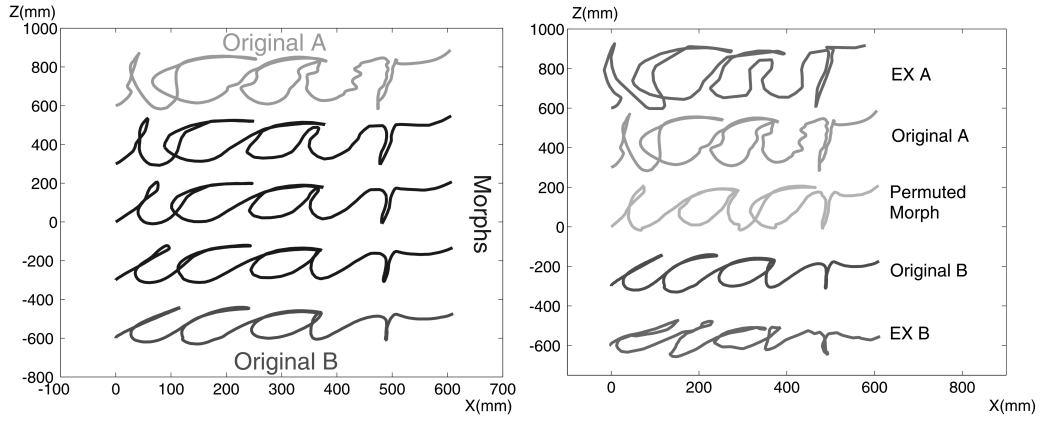


Figure 5.5: Left panel: Recorded pen trajectories and synthesized movements. The morphs interpolate continuously in space-time between the prototypes. Right panel: Original pen trajectories and exaggerations of the writing styles of the two actors. The middle row shows synthesis of a new word "IACR" by reassembling the movement primitives in a different sequential order.

Transfer to the Robot arm: The synthesized movements were executed using a Mitsubishi PA-10 7-DOF robot arm (fig. 5.6). Optimization has been performed for different values of α (see section. 5.3.3). Figure 5.6 illustrates that for small values of α a better imitation (measured by the difference $\|\mathbf{e}_d - \mathbf{e}_d\|$) is achieved but discontinuous joint trajectories can arise. These discontinuities disappear for large values of α at the cost of worse imitation quality.

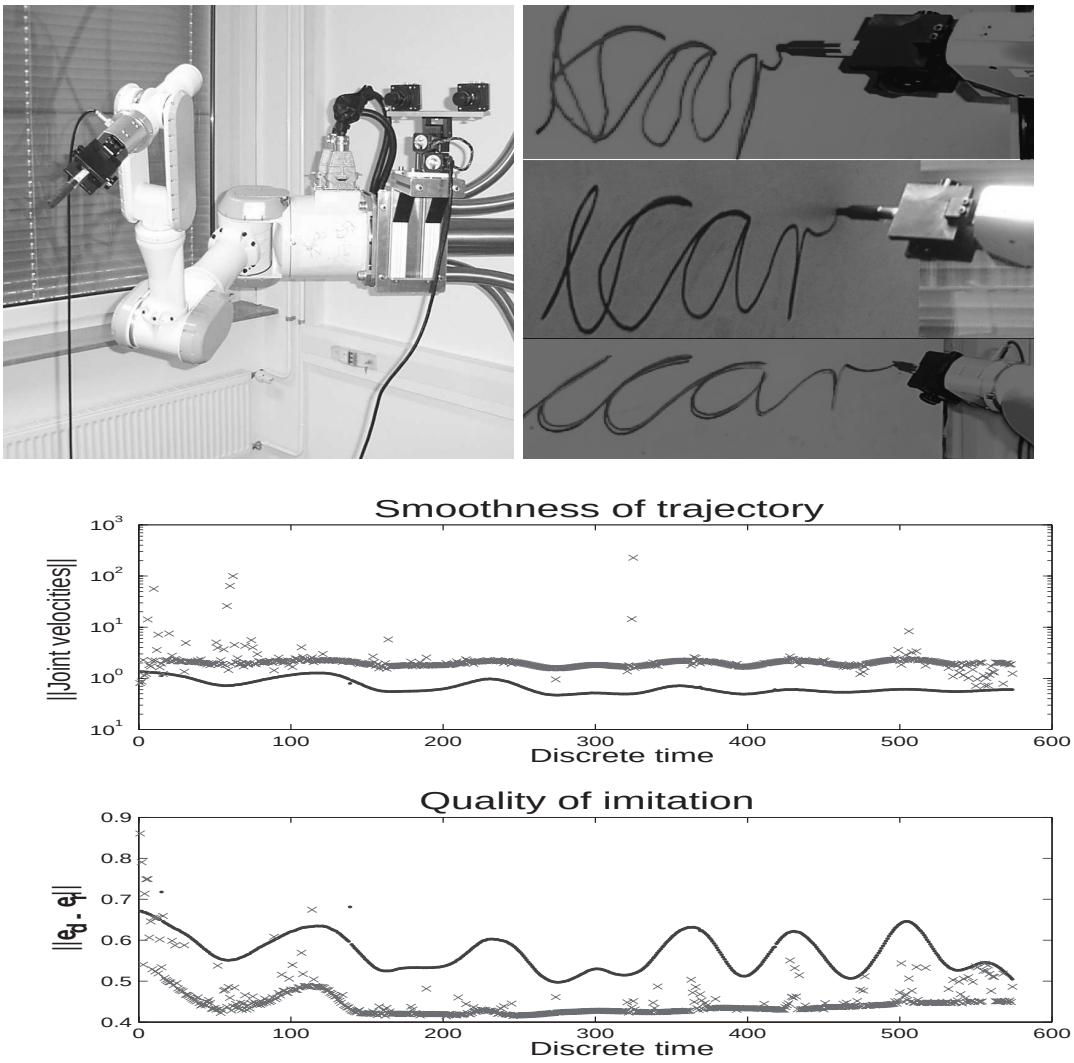


Figure 5.6: Top-Left: The Mitsubishi PA–10 robot arm used to execute the writing movements. Top-Right: Writing examples of the Originals A and B and the average morph in between (compare fig. 5.5). Bottom: Joint velocities $\|\mathbf{q}(t_i) - \mathbf{q}(t_{i-1})\|$ as a function of time. For $\alpha = 10^{-1}$ (dots) and $\alpha = 10^{-2}$ (crosses). One obtains continuous joint trajectories for larger α.

5.5 Should a robot imitate?

In this section we investigate the effect of imitating a human posture on various dexterity measures. Dexterity measures assess the quality of a posture with regards to specific aspects such as flexibility, safety and energy consumption. We analyze the differences between imitated trajectories and trajectories generated by a standard controller as delivered by the manufacturer and discuss the effect of imitating movements from the engineering viewpoint.

Joint Limits The distance from mechanical joint limits $q_{i_{min}}$,$q_{i_{max}}$, can be defined by the vector

$$\mathbf{e}_q = \left[\frac{1}{\Delta_1}(q_1 - \frac{\Delta_1}{2}), \dots \frac{1}{\Delta_7}(q_7 - \frac{\Delta_7}{2}) \right], \tag{11}$$

where $\Delta_i = q_{i_{max}} - q_{i_{min}}$. In general, the bigger the distance from a mechanical joint limit, the more agile is the robot. Figure 5.7 shows the overall distances from mechanical joint limits for trajectories generated by imitation. One can see that the robot controller typically uses joint values closer to the limits. This seems to be surprising at first sight. However, it is likely to be a task-specific phenomenon since humans and robots use quite completely different control policies.

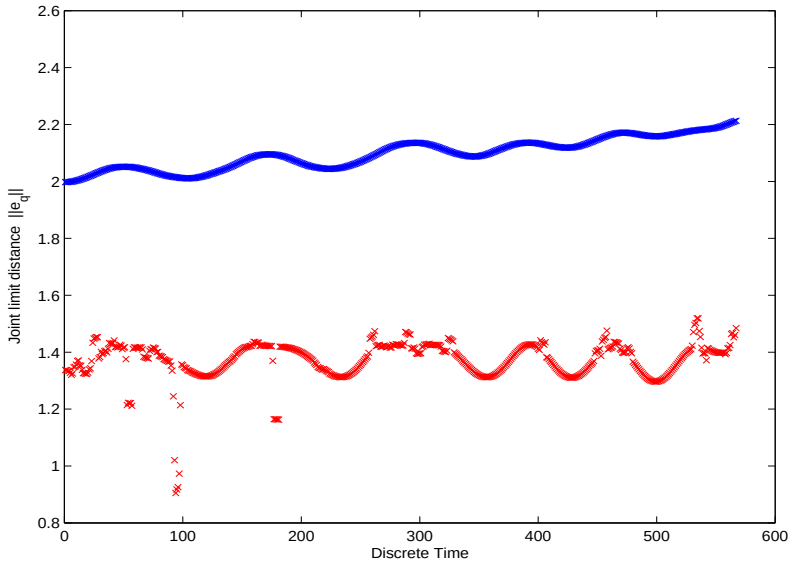


Figure 5.7: The overall distance of joint trajectories for imitating (lower curve) and standard trajectories (upper curve). The standard controller generates joint trajectories which are closer to the joint limits.

Distance from Singularity For any robot one can relate the joint values $\mathbf{q}(t_i)$ with the Cartesian task coordinates of the final tool tip via the so called forward kinematics map:

$$\zeta(t_i) = \mathbf{P}_r(\mathbf{q}(t_i)). \tag{12}$$

Taking the derivative of (12) with respect to t_i yields:

$$\dot{\zeta}(t_i) = \mathbf{J}(\mathbf{q}(t_i))\dot{\mathbf{q}}(t_i), \tag{13}$$

where the Jacobian $\mathbf{J}$ relates joint velocities to robot velocities in the task space. As proposed in [103] and [51], the distance from a singularity, i.e. a configuration where the robot looses at least one degree of freedom, can be characterized by the minimum singular value σ_{min} of the Jacobian matrix $\mathbf{J}$. The trajectory of the minimum singular value for both trajectories is shown in figure 5.8. One can see from figure 5.8 that the standard controller generates trajectories which are more *stable* and never causing critical movements. In contrast, this frequently happens for the imitated trajectories.

Manipulability Although a robot does not loose any degree of freedom it might be strongly constrained in the possible movements it is allowed to do. A measure for *manipulability* was introduced in [75] and consists of the quotient between largest and smallest singular value of the Jacobian matrix, i.e.: $c_{\mathbf{J}} = \frac{\sigma_{max}}{\sigma_{min}}$. In figure 5.9 we show the condition number for both trajectories evaluated at each time point t_i. Note that the trajectory of

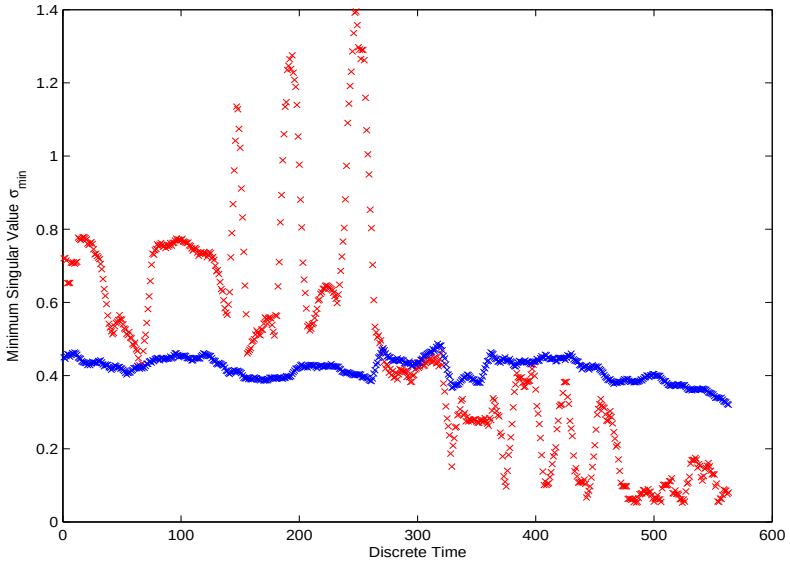


Figure 5.8: Minimum singular values σ_{min} for trajectory generated by standard controller (dark) and imitated robot trajectory(light)

the standard controller is almost constant across the whole movement, whereas the imitated trajectory leads to a strong change of the manipulability measure. Is is reasonable to assume that the robot controller has some form of minimum manipulability constraint which is considered during the whole motion. Since the robot has a redundant degree of freedom, this redundant degree of freedom might be used to keep the manipulability constant. However, the trajectory generated by imitation is using this degree of freedom for imitation. Thus the only way to increase the manipulability during imitation would be a make a trade-off between imitation and manipulability.

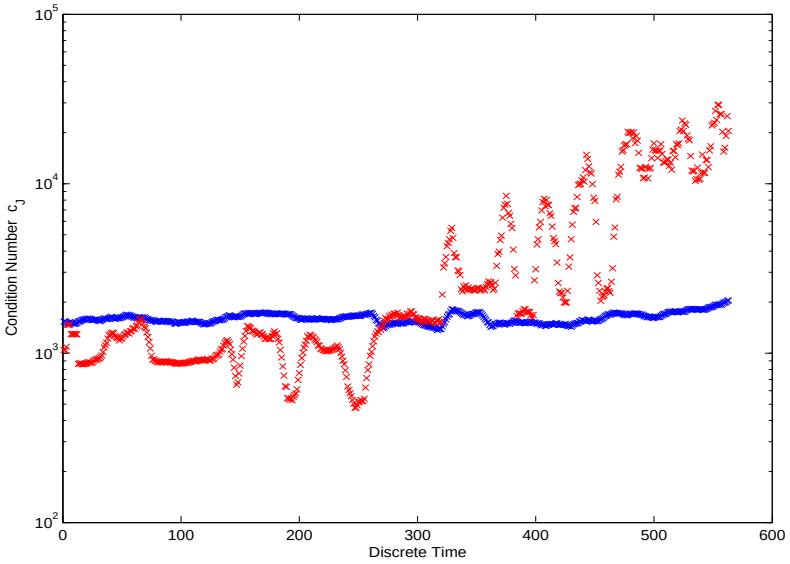


Figure 5.9: Ratio between maximum and minimum singular values of the Jacobian matrix for trajectories generated by the standard controller (dark) and imitated robot trajectory(light)

Kinetic Energy According to [84] the kinetic energy K of a manipulator can be expressed as

$$K = \frac{1}{2}\mathbf{v}^T\mathbf{D}(t_i)\mathbf{v}, \tag{14}$$

where $\mathbf{D}(t_i)$ is the inertia matrix depending on the actual posture and $\mathbf{v} = \dot{\mathbf{q}}$. Since $\mathbf{D}$ is always positive definite K increases with $\|\mathbf{v}\|$. The higher the overall joint velocity the higher is the kinetic energy of the manipulator, which leads to higher motor torques and therefore higher energy consumption. We show the joint velocities of both trajectories in figure 5.10. Note the similarities between both curves. Assume that we perform a small movement in Cartesian space $\boldsymbol{\delta\zeta}$. Using the Jacobian we obtain for the implied velocity $\boldsymbol{\delta\mathbf{q}}$ the relation

$$\boldsymbol{\delta\mathbf{q}} = \mathbf{J}^{-1}\boldsymbol{\delta\zeta}.$$

Now, the link is not so surprising since we see that if a trajectory passes nearby a singularity the Jacobian gets worse conditioned and therefore a small movement in Cartesian space has to lead to large joint velocities and therefore also to higher kinetic energy.

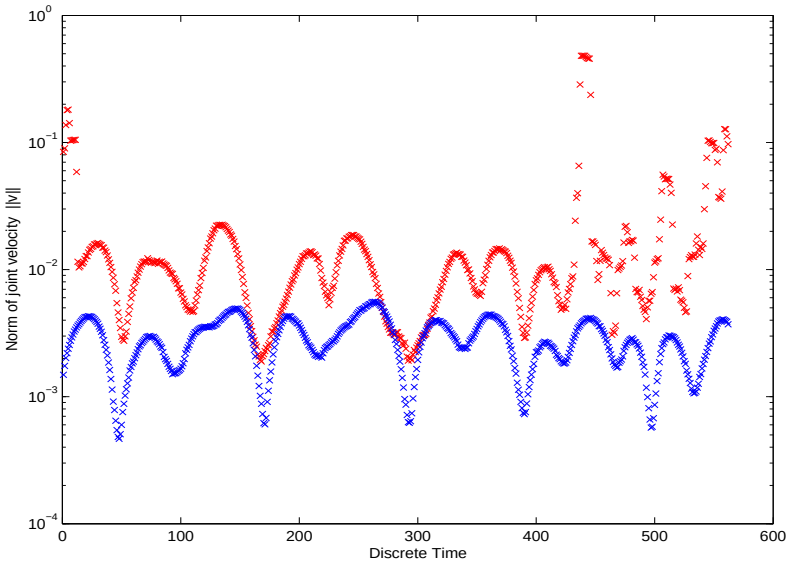


Figure 5.10: Norm of joint velocity vector $\mathbf{v}$ generated by standard controller (dark) and imitating robot trajectory(light).

5.6 Conclusion

We have presented a method for imitation learning of complex movement trajectories that is based on linear combination of small sets of prototypical example movement sequences. The proposed algorithm decomposes long trajectories automatically into movement primitives, and models these primitives by linear combination of prototypical trajectories. We also have shown how such flexible representations of movement trajectories can be coupled with a real robot system in a way that ensures the accurate reproduction of endpoint trajectories and the imitation of the style of the human movement. The proposed method for transferring the synthesized trajectories to the robot has the advantage that it combines an *exact* control of the end-effector position with a "softer" posture control that characterize the style of the executed arm movements. Since the current formulation aims at

exact task execution and soft imitation, the task transfer is an exact optimization problem since no compromise on the accuracy in task execution is desired. This prohibits the use of a learning scheme for this task.

We also discussed the properties of the robot imitating trajectories from a dexterity analysis point of view. We found that trajectories performed by humans are not necessarily the best trajectories to achieve a task for a robot. For example, trajectories obtained by a standard controller will use the degree of freedom in the arm to increase manipulability and try to decrease kinetic energy of the arm. In contrast, ensuring accurate reproduction of endpoint trajectories, the imitation of the style of the human movement will explore this degree of freedom to imitate and generate trajectories which are therefore suboptimal in the robotic sense. An interesting fact is that humans take advantage of posture singularities while robots do not. For example, during a lifting process, humans use the mechanical limit of the joints to block their own motion and thus increase their arm stiffness.

In summary, we conclude that a robot controller which imitates humans has to find a suitable trade-off between accuracy of imitation and penalizing postures that are near singularities. This trade-off could done for example by adding penalty terms to the optimization procedure. Our approach to imitation is purely optimization-based. Thus, a crucial assumption is that we know the trajectory to be imitated *precisely*. However, since we now have seen that *perfect* imitation may not be desirable anyway, we will investigate ways to relax the assumption of perfect environmental knowledge in the next chapter.

Chapter 6

Robot Imitation – A Learning based Approach

In chapter 5 we have discussed the problem of transferring perceived movement characteristics to a robot arm. In particular, we found out that this approach suffers from the following disadvantages:

- The robot imitates the human actor *after* performing a computational expensive optimization. As formulated in chapter 5, imitation is a very time consuming and therefore inconvenient teach-in mechanism.
- Imitation works only under ideal conditions: movement characteristics were required to be given by a complete set of 3D world coordinates.
- Imitation of posture is a suboptimal control policy when performed for each single point of the trajectory.

In contrast, in this chapter we propose a learning-based approach to imitation: Given a picture of a human actor as input, what is the corresponding robot posture? Therefore,

- we tackle the imitation problem when perception is not god-like: We do not need any 3D coordinates.
- we only want to generate control points of a trajectory. Thus the robot can use a different control policy for trajectory interpolation.
- imitation is formulated as an estimation task which requires predictions only during imitation. It will turn out that this can be done more efficiently than in the optimization-based approach.

Our approach requires us to solve the problem of pose recognition *and* simultaneously the problem of estimating the output robot posture from a given set of image features. For the solution of these problems we will mainly use ideas from Kernel Dependency Estimation. For the KDE map we use image features encoding the pose of the human actor as inputs and the corresponding robot posture as desired output (see figure 6.1). Note that this is not a standard regression problem, since there exist a non-linear dependence of the outputs due to the kinematics of the robot arm. Of course, by using unreliable information about the imitated trajectories we have to sacrifice precision. Thus we cannot test the new approach in the the same task as in chapter 5. We discuss in section 6.1 problems involved in estimation of human posture given an image $\mathcal{I}$ with a human actor. Note that since

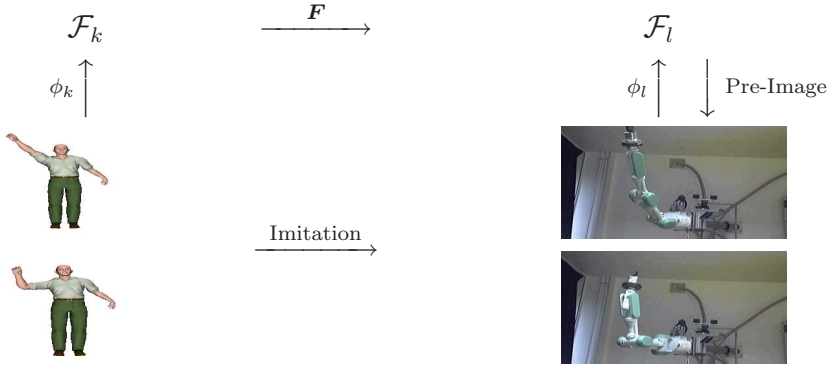


Figure 6.1: KDE for robot imitation.

our ultimate goal is to estimate the robot posture given an input image we do not need to estimate any kinematic model of a human as intermediate step. Posing the whole problem as a single inference problem allows us to circumvent the usage of additional models. Finally, note that a valid robot joint posture has to obey the constraints of kinematic and dynamic feasibility. Therefore, we discuss the inference problem of robot joint variables given a set of input features under these additional constraints in section 6.2.2. We will encode the additional kinematic constraints as constraints for the pre-image problem in KDE. Let us start in the next section with the first problem: Pose Estimation.

6.1 Estimation of Human Pose

Human pose estimation has received a considerable interest in the last decade due its importance for man-machine interaction. Existing algorithms found in the literature can be categorized into two categories:

- Model-based algorithms which formulate the pose estimation as estimation problem of parameters of an a-priori defined articulated structure which is observed in the image.

- Template-based algorithms which extract a set of possible elementary image features (as landmarks and corner/color histogram) from an image and then try to find most the likely posture by matching features of a template.

Algorithms based on an a priori defined kinematic model of a human body usually aim at solving an incremental tracking problem. Such algorithms try to estimate the current state of the kinematic model parameters given the image as observation and the current state of the kinematic model. For an implementation of such systems consider e.g. [27], [9] and [8]. Since we are not interested in a kinematic model of a human, we do not consider model based approaches, but merely focus in the following on the template based approach. Furthermore, note that our aim is to use a single image source only. Thus we do not consider approaches using multiple views to resolve ambiguities or to estimate 3D information directly from the image such as in [62] or [30].

We start with an approach based on *contours* introduced by [47] and which was used for pose estimation in [1]. The contour-based approach is particular interesting since it creates features which encode *global* properties of the pose. Unfortunately, contour-based

approaches are sensitive to arm-body self-occlusion (see discussion below). To this end, we investigate an alternative approach based on skin-color which uses color information to determine arm,hand and head position. While color is not sensitive to self-occlusion it is a source of trouble on its own since it depends on the lighting condition and texture information which both can vary dramatically. Finally, in our approach, we investigate a combination of both approaches by fusing cues from global shape with color information.

6.1.1 Contour descriptors

The basic assumption of contour descriptors is that the information of the object of interest can be represented by a planar and possibly closed curve. Consider the contours of human poses shown in figure 6.2. As a human observer, just looking at the contour is sufficient to estimate the pose of the human. Therefore, the contour is a rich descriptor which could be used to measure pose. Thus we look for features which would allow a machine to measure

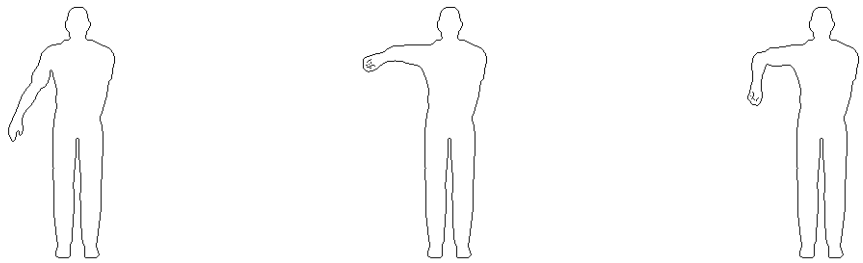


Figure 6.2: The contour of an object can be sufficient to determine its pose.

similarities between contours and thus to estimate pose by applying a learning algorithm based on such a similarity measure. To this end, one has to first extract contours from an image and then solve the problem of finding a representation of the contour which allows to define some sort of distance measure on contours. Let us discuss in the following how to extract contours first.

A contour is intuitively a curve which delimits and thus is the boundary line of different image regions. For example, given a segmented image, the contour of an object can be the delimiter to the background. In a color image the regions of similar color can be a separated by a contour. In general the points which belong to a contour are edges in an image and thus we will use standard edge extraction algorithms to calculate the edge map $\mathcal{J}$, where $\mathcal{J}$ is of the same size as the original image $\mathcal{I}$ and where

$$\mathcal{J}(x,y) = \begin{cases} 1 & \text{if} \quad \text{edge found at } \mathcal{I}(x,y) \\ & 0 \quad \text{otherwise} \end{cases}$$

Such edge maps can be obtained by calculating the image gradient. This operation can be efficiently implemented using a linear convolution of the image with a so-called Sobel filter(see [68] for details). For example the contours in figure 6.2 are obtained by this procedure. In general, before convolving the image with a Sobel filter it is useful to smooth the image to remove any possible spurious edges. Such smoothing can also be realized by an efficient convolution operation. After obtaining the edge map $\mathcal{J}$ one has to choose a parametrization of the contour which ideally has the following properties:

- Scale invariance: the pose is not dependent on the distance of the human in the image.
- Translation invariance: the pose is not dependent on the relative position of the human to the robot.
- Continuity: Small change in pose should result in a small change of the features.

Obviously, we cannot use the edge map $\mathcal{J}$ directly as a representation since it does not have any of the 3 properties above. Therefore, we have to consider an alternative representation which have the desired properties. In this thesis we will consider Shape-Contexts.

Shape-Context The parametric form of the contour allows us to compare two contours via their L_2 distance. However, if our main interest is just comparing contours and not visualizion we can pursue a different approach. In [3] it was proposed to use *shape-context* descriptors which are local radial histograms located at points on the edge map and which are calculated over local neighborhoods. Such shape-contexts were used in [64] for human pose estimation and in [1] for contour-based tracking of human pose. Let us review the shape context approach in more detail. We start again with the edge map $\mathcal{J}$ obtained by applying a standard edge detector to the segmented image $\mathcal{I}$. Now, the next step is to select r points from the edge map $\mathcal{J}$ which are likely to be part of the contour. To this end, given all pixel coordinates $E := \{(i,j)\}$ of pixels where $\mathcal{J}(i,j) = 1$, so called edgels, we construct the convex hull $C(E)$ of all edgels and randomly select r edgels which are close to the convex hull to select a subset of all edgels to keep processing requirements low. Let the selected subset of edgels be denoted as $\bar{E} \subset E$. Given this r edgels we can calculate the r^2 pixel distances among each edgel e_i, e_j

$$D_{ij} = ||e_i - e_j||^2,$$

and the mean distance m_D via

$$m_D = \frac{1}{r^2} \sum_{i,j}^{r} D_{ij},$$

which allows us now to rescale all selected edges in $\bar{E}$ and thus leads to removing the scale information. In the next step we select an edgel e_i in $\bar{E}$ and use it to center all other edgels in $\bar{E} \setminus \{e_i\}$. The new set of centered edgels are now used to calculate the log polar histogram at this particular edgel e_i. To this end, we first introduce the two normals w_1 and w_2 which define two lines which cross at zero and which have a phase difference of $\delta\phi$. Rotating w_1 and w_2 allows us to define a one-dimensional histogram $h(\theta)$ over the rotating angle θ via

$$h_i(\theta) = \#\{e_j | w_2^\top e_j < 0 \text{ and } w_1^\top e_j \geq 0\} \quad \text{with } e_j \in \bar{E} \setminus \{e_i\}.$$

We can extend this one-dimensional histogram by introducing *distance levels* which introduce further bins and which aim to count the number of edgels which are at a particular distance from the central edgel. To emphasize the vicinity, the distance levels are uniform at log scale which implies that there are more bins in the vicinity of the center edgel e_i. Thus our final radial log polar histogram looks like

$$h_i(\theta, k) = \#\{e_j | w_2^\top e_j < 0 \text{ and } w_1^\top e_j \geq 0 \text{ and } d_k < ||e_i - e_j||^2 < d_{k+1}\} \quad \text{with } e_j \in \bar{E} \setminus \{e_i\},$$

where d_k is a sequence of thresholds which are required to be equidistant on log scale (for example $d_k = \exp(\gamma k)$ with $\gamma > 0$). Note that the two-dimensional histogram can be

represented as a matrix H_i. In figure 6.3 we visualize the radial log polar histogram where we have used $\delta\phi = \phi/5$ and $\theta = (0, \pi/5, \ldots, 2\pi)$ as binning values. The above procedure is repeated for r edgels and yields a set of shape descriptors $S = \{(e_1, H_1), \ldots, (e_r, H_r)\}$. In [3] it is argued that this set of shape-contexts is a rich descriptor and preserves the exact shape information as long the bins are fine enough.

In this thesis we will construct features based on shape-contexts that will serve us as input features for the regression algorithm to estimate the output pose. However, since contours are usually not robust and are sensitive to self-occlusion, we describe in the following a color-based approach. Color will serve us as an additional input cue and we aim at combining both types of features for human pose estimation. The motivation for combining features is that each of the single features might be too less discriminative and thus using a richer set on features should yield more discriminative power

6.1.2 Skin-Color based pose estimators

The motivation to use the color of skin as feature for human pose estimation is based on a number of facts related to skin color. First, obviously color processing is very efficient since only three numbers (red, green and blue intensity – RGB) are involved. Therefore we do not need to perform any convolution operations as in the contour-based approach. Second, assuming arms and faces can be observed unclothed, the regions of skin color are a strong indicator of the human pose. Most approaches try to model the variety of skin colors using a probabilistic model obtained from example images containing skin color. Let us denote the RGB color value at pixel (i,j) by $\mathrm{rgb}(i, j)$. The probability of a color value $\mathrm{rgb}(x)$ at pixel coordinate $x = (i, j)$ being skin is then

$$p_{\mathrm{skin}}(\mathrm{rgb}(x)|\eta),$$

which can be given by e.g. a single Gaussian in the three-dimensional RGB space. Here, η indicates the necessary parameters. In the case of a Gaussian it would consist of the mean and the covariance. In this thesis, we use a single Gaussian distribution function to model the distribution of skin color values. Thus in our case p_{skin} is a multivariate Gaussian in a three-dimensional space. We determine the parameters η of our Gaussian distribution function by collecting a set of camera images and choosing manually pixels which belong to skin. Afterwords, we calculate the mean RGB color value and the covariance of all

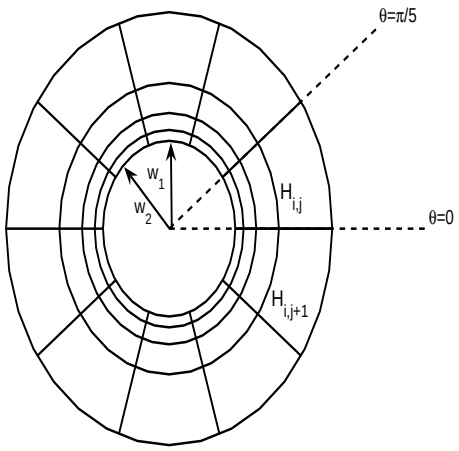


Figure 6.3: Anatomy of radial log polar histograms. See text for description.

collected color values. A more sophisticated approach is implemented in [45] which used a mixture of Gaussian to model the probability p_{skin}. In this case η defines a whole set of mean vectors and covariance matrices.

We can now use the probability function p_{skin} to calculate the skin color probability of each pixel in a new image to obtain a new image $\mathcal{S}$ where each pixel given by

$$\mathcal{S}(i,j) = p_{\text{skin}}(\text{rgb}(i,j)|\eta).$$

Thus the brightness of each pixel in $\mathcal{S}(i,j)$ indicates the probability of this pixel to belong to skin. We will denote $\mathcal{S}$ as the *skin map*. After obtaining the skin map, we need useful features for extracting the pose. Assume that the image is segmented and we only evaluate the probability on pixels belonging to the human actor in a bounding box indicating the region of interest. Then, we can calculate two histograms $h_x(j), h_y(i)$ which, given a threshold value σ, encode the distribution of skin colors in the bounding box via

$$\begin{aligned} h_x(j) &= \#\{i|p_{\text{skin}}(\text{rgb}(i,j)|\eta) > \sigma\}, \\ h_y(i) &= \#\{j|p_{\text{skin}}(\text{rgb}(i,j)|\eta) > \sigma\}. \end{aligned}$$

Concatenating these two histograms to a single feature vector gives a simple representation of the distribution of skin and thus the human pose. In general, it is beneficial to smooth the image before calculating the skin map and the histograms. Furthermore, one can perform a dilation operation optionally to get connected areas in the skin map [68]. In figure 6.4 we show the evolution of the histograms h_x and h_y while a human actor waves his right arm.

6.1.3 Combining color and shape cues

We combine shape and color features to build a single feature vector that contains sufficient information about the posture. Let us introduce the two feature spaces $\mathcal{F}_S, \mathcal{F}_C$ where $\mathcal{F}_S$ is the feature space constructed from the shape descriptors and $\mathcal{F}_C$ is the feature space constructed using color histograms h_x and h_y. Thus we consider the feature vector $\boldsymbol{x} = [\phi(S(\mathcal{I}), \phi([h_x(\mathcal{I}), h_y(\mathcal{I})])] \in \mathcal{F}_S \times \mathcal{F}_C$ where $S(\mathcal{I})$ is a shape descriptor and $h_x(\mathcal{I}), h_y(\mathcal{I})$ are the concatenated color histograms of image $\mathcal{I}$. A dot product, and thus a kernel k_{s-c} between two such feature vectors obtained from two images $\mathcal{I}_1, \mathcal{I}_2$ could be implemented by the sum of two kernel functions

$$\begin{aligned} k_{s-c}(\mathcal{I}_1, \mathcal{I}_2) = \boldsymbol{x}^\top \boldsymbol{y} &= \phi(S(\mathcal{I}_1)^\top \phi(S(\mathcal{I}_2)) + \phi([h_x(\mathcal{I}_1), h_y(\mathcal{I}_1)])^\top \phi([h_x(\mathcal{I}_2), h_y(\mathcal{I}_2)] \quad (1) \\ &=: k_S(\mathcal{I}_1, \mathcal{I}_2) + k_C(\mathcal{I}_1, \mathcal{I}_2). \quad (2) \end{aligned}$$

Here, the kernel function k_S is a similarity measure between the shape descriptors of images $\mathcal{I}_1$ and $\mathcal{I}_2$. Analogously k_C is a similarity measure between the color histograms of $\mathcal{I}_1$ and $\mathcal{I}_2$. Let us start with the description of the color histogram kernel $k_C(\mathcal{I}_1, \mathcal{I}_2)$. In [36], kernels between probability distributions are discussed. They argued not to use a linear kernel function to compare histograms since a linear kernel, i.e. the euclidian dot product, does not consider special properties of histograms as binning or quantization. As alernative they discussed several kernel functions which are expected to outperform linear kernels on histograms. In our work we use their "total variation" kernel $k_{\infty|1}$ since it is almost as efficient to calculate as the linear kernel function. The total variation kernel is given by

$$k_{\infty|1}(p,q) = \sum_{i=1}^{N} min(p_i, q_i),$$

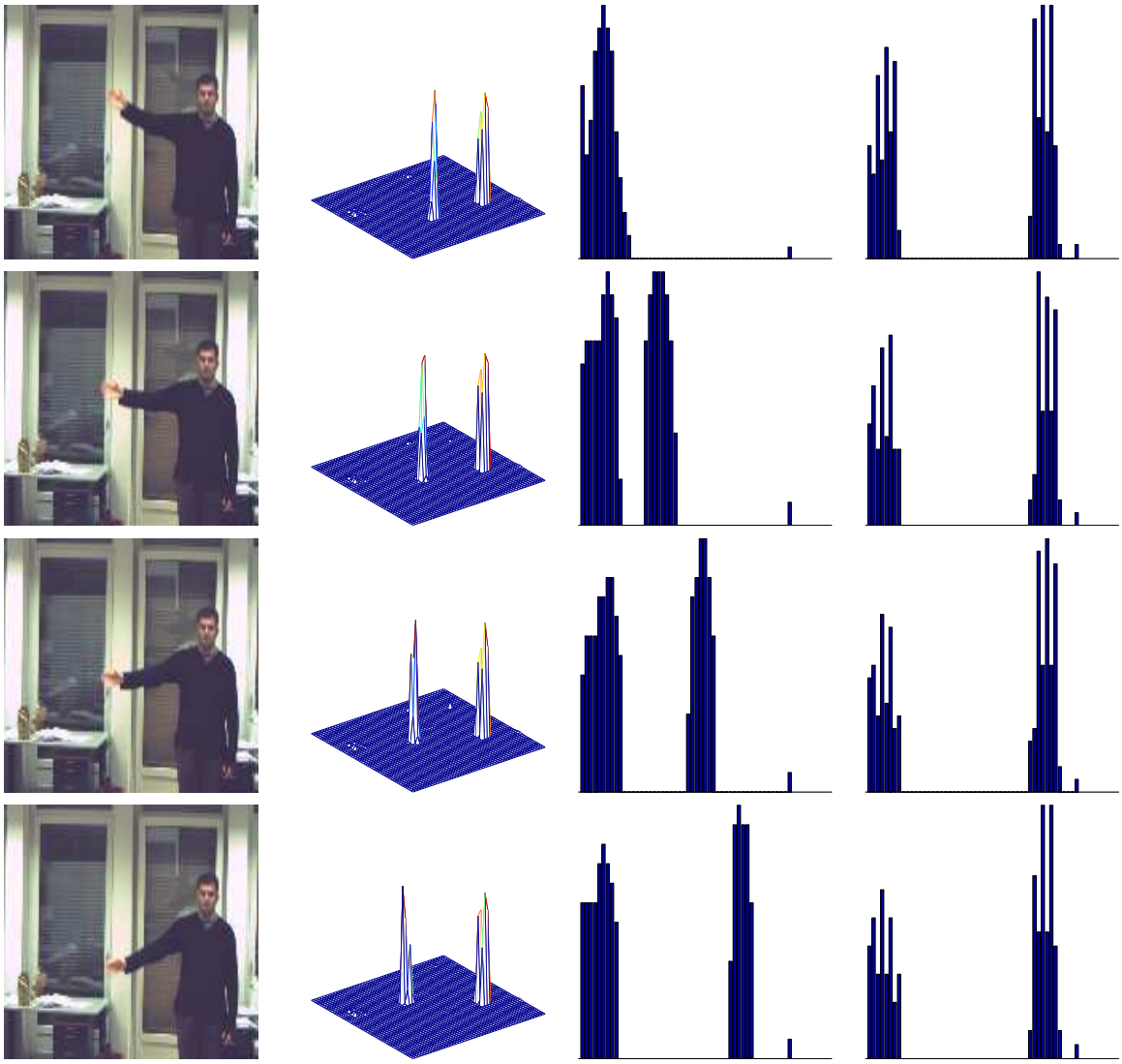


Figure 6.4: Skin Color based features of human pose. Column 1: Camera Image; 2: Skin Map $\mathcal{S}$; 3: Histogram of row response $h_x(j)$; 4: Histogram of column response $h_y(i)$.

for p, q being histograms with N bins. Since our kernel k_C is the similarity between a pair of histograms we use the total-variation kernel for each row and column histogram, i.e. h_x and h_y separately and sum the result. Thus our k_C kernel takes the form

$$k_C(\mathcal{I}_1, \mathcal{I}_2) = k_{\infty|1}(h_x(\mathcal{I}_1), h_x(\mathcal{I}_2)) + k_{\infty|1}(h_y(\mathcal{I}_1), h_y(\mathcal{I}_2)).$$

To construct the kernel k_S between two shape descriptors $S(\mathcal{I}_1), S(\mathcal{I}_2)$ we pursue a different direction. Since $S(\mathcal{I}_1), S(\mathcal{I}_2)$ are sets with a different number of elements s_i we would have to define a kernel which measures similarity between two sets. There exist kernels for sets, e.g. [53], but they are typically expensive to compute. Therefore, we follow an approach by [17] where the authors propose to reduce the set to the *most relevant* subset $\hat{S} \in S$. Elements of the subset $\hat{S}$ can consist, for example, of cluster centers obtained from clustering elements in S, or of codebook vectors obtained by performing a vector quantization. We use vector quantization as proposed by [91] which requires us to specify a distance measure $d(s_i, s_j)$ between shape contexts s_i and s_j. Remember that each

$s_i \in S$ was a pair (e, H), where e was an edgel coordinate and $H \in \mathbb{R}^{m \times n}$ a radial log polar histogram. Since we are not interested in absolute coordinates we will only consider the histogram H. Furthermore we will use the distance measure associated with the total-variation kernel $k_{\infty|1}$ which we already have used for histogram comparison. Thus the distance between shape contexts is given by[1]

$$\begin{aligned} d(s_i, s_j)^2 &= k_{\infty|1}(H(s_1), H(s_1)) + k_{\infty|1}(H(s_2), H(s_2)) - 2k_{\infty|1}(H(s_1), H(s_2)), \quad (3) \\ &= \mathbf{1}_m^\top H(s_1)\mathbf{1}_n + \mathbf{1}_m^\top H(s_2)\mathbf{1}_n - 2k_{\infty|1}(H(s_1), H(s_2)), \quad (4) \end{aligned}$$

where $\mathbf{1}_m, \mathbf{1}_n$ are vectors of m and n ones respectively. We have used the fact that

$$k_{\infty|1}(H, H) = \sum_{i=1}^{m}\sum_{j=1}^{n} \min(H_{ij}, H_{ij}) = \mathbf{1}_m^\top H \mathbf{1}_n.$$

If we consider multiple contours at ones, we can collect a very large set S of shape contexts and thus are ensured to observe a large variety of possible shape contexts. Applying vector quantization to this set will result in a subset $\hat{S}$ of shape contexts which *cover* all shape contexts in S sufficiently well. Once $\hat{S}$ is determined, it can be used to extract features of fixed size for a new image. Let $|\hat{S}|$ denote the number of obtained codebook vectors. Given a new image $\mathcal{I}$ we first extract shape descriptors and obtain $S(\mathcal{I})$ as described above. Then, we can use the codebook vectors in $\hat{S}$ to calculate a histogram h_S where the i-th bin counts the occurrence of codebook vector $s_i \in \hat{S}$ in the set of shape contexts $S(\mathcal{I})$, i.e.:

$$h_S(i) = \#\{s \in S(\mathcal{I}) | i = \arg\min_{s_i \in \hat{S}} d(s_i, s)\}.$$

Note that h_S has as many bins as codebook vectors in $|\hat{S}|$ independent of the number of found edges (and thus shape contexts) in the image $\mathcal{I}$. Thus our similarity measure k_S between the two sets $S(\mathcal{I}_1)$, $S(\mathcal{I}_2)$ consists of calculating the two histograms $h_S(\mathcal{I}_1)$,$h_S(\mathcal{I}_2)$ first and then measuring the similarity on the resulting histograms via the total-variation kernel. Now the input features and the corresponding similarity measures for our KDE approach are defined. Let us discuss in the following section the output side: features and kernels for robot posture.

6.2 From Human to Robot Posture

The robot posture is fully specified by the seven joint values of the robot arm. Thus, elements of our output set $\mathcal{Y}$ are elements of a seven-dimensional vector space. However, the feasible set of joint values is highly correlated due to the kinematic constraints of the robot. Indeed $\mathcal{Y}$ is a nonlinear manifold in $\mathbb{R}^7$. The classical approach would try to model the manifold explicitly. However, since we are only interested in a submanifold, the submanifold of human like poses, we are unable to model this submanifold explicitly. Fortunately we are able to observe points from this submanifold and thus utilize a learning approach such as KDE. To do so, in the next section we define our output kernel which measures similarities among the seven-dimensional vectors of joint values which takes into account the kinematic relationship among joints. After estimation we need to solve the pre-image problem. In our case the pre-image problem is used to find the target joint values of the robot. In contrast to the introduced pre-image techniques we will use

[1] Note that the squared distance between points in a linear space can be expressed by evaluating dot-products only.

intermediate solutions of the pre-image problem to construct intermediate joint values and thus automatically interpolate from an initial set of joint values to target joint values. This is necessary since the transition of robot states *between* poses must be smoothly.

6.2.1 A similarity measure on kinematic chains.

Let us define the vector $\boldsymbol{q} = [q_1, \ldots, q_7] \in \mathbb{R}^7$ where each entry q_i defines the value of joint number i. We assume that q_1 is the base joint and larger joint numbers indicate that a joint is closer to the wrist. Let us briefly introduce the notion of *forward* kinematics which we will need to describe the relationship between joint values and position of each robot link and orientation in Cartesian space. Consider the forward kinematic map

$$F_k(\boldsymbol{q}) = \prod_{i=1}^{k} A_i(q_i), \tag{5}$$

where A_i is typically a 4 by 4 transformation matrix which maps homogeneous coordinates of points in frame $i-1$ to coordinates in frame i. In general $A_i(q_i)$ is provided by the robot manufacturer. Assuming that the Denavit-Hartenberg [83] convention is used to describe the kinematic parameters of the robot arm, the matrix A_i has the following structure:

$$\begin{aligned} A_i(q_i) &= \begin{bmatrix} \overbrace{\text{Relative Rotation}}^{3\times 3} & \overbrace{\text{Relative displacement}}^{3\times 1} \\ \mathbf{0}^\top & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos(q_i) & -\sin(q_i)\cos(\alpha_i) & \sin(q_i)\sin(\alpha_i) & a_i\cos(q_i) \\ \sin(q_i) & \cos(q_i)\cos(\alpha_i) & -\cos(q_i)\sin(\alpha_i) & a_i\sin(q_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}, \end{aligned}$$

where α_i, d_i, a_i are constant values of each link and describe quantities such as length, relative rotation and orthogonal displacement to link $i-1$. As one can easily see, the transformation F_k is nonlinear and maps a set of joint values from the joint space via a set of trigonometric equations to a 4×4 matrix $F_k(\boldsymbol{q})$. Let us denote this particular matrix $F_k(\boldsymbol{q})$ by T_0^k. From the anatomy of A_i one can see that all orientation information must be stored in the upper left 3×3 submatrix R_k of T_0^k and likewise all position information p_k is stored in the last column of T_0^k, thus T_0^k has the structure

$$T_0^k = \begin{bmatrix} \mathbb{R}_k & p_k \\ \mathbf{0} & 1 \end{bmatrix}.$$

The pose of the arm is fully determined by the position information given by p_1 to p_7. Furtermore, to simplify the notation let us denote by $p_k(\boldsymbol{q})$ the position vector p_k which we obtain by extracting the last column of $T_0^k = F_k(\boldsymbol{q})$.

Let us now come back to our main interest: the definition of an appropriate similarity measure between two joint vectors. Let us begin with arguing why the standard Euclidean measure is not appropriate for measuring similarities among joint values. Given two joint vectors $\boldsymbol{q}^1$ and $\boldsymbol{q}^2$ the standard Euclidian dot product is:

$$k_{lin}(\boldsymbol{q}^1, \boldsymbol{q}^2) = \sum_{i=1} q_i^1 q_i^2.$$

Let us perturb now a single entry q_i^1 by a small value ϵ. It is clear that k_{lin} will change by $O(\epsilon)$, but this small change will have an effect on all points $p_j(\boldsymbol{q}^1)$ with $j \geq i$ through

the nonlinearities of the forward kinematic map. Therefore, the resulting pose of joint values $\boldsymbol{q}^1$ and $\boldsymbol{q}^2$ can vary dramatically. Thus the standard inner product k_{lin} is not an appropriate similarity measure of pose. A more suitable similarity measure could be the three-dimensional vectors $p_i(\boldsymbol{q})$. We propose to use the sequence of vectors

$$[p_1(\boldsymbol{q}^1), p_2(\boldsymbol{q}^1), \ldots, p_7(\boldsymbol{q}^1)], [p_1(\boldsymbol{q}^2), p_2(\boldsymbol{q}^2), \ldots, p_7(\boldsymbol{q}^2)]$$

to calculate the weighted *pose kernel*

$$k_{pose}(\boldsymbol{q}^1, \boldsymbol{q}^2) = \sum_{i=1}^{7} \lambda_i k(p_i(\boldsymbol{q}^1), p_i(\boldsymbol{q}^2)), \tag{6}$$

which require a base kernel $k : \mathbb{R}^3 \times \mathbb{R}^3 \to \mathbb{R}$ to measure the similarity between the Euclidean position of the robot link i with joint values $\boldsymbol{q}^1$ and joint values $\boldsymbol{q}^2$. Note that this pose kernel corresponds to a feature space $\mathcal{F}_1 \times \cdot \times \mathcal{F}_7$ where each $\mathcal{F}_i$ is constructed using the base kernel $\lambda_i k(\cdot, \cdot)$ and the Euclidean coordinates of joint i. In this thesis we use a linear kernel function as base kernel. A nice property of this kernel is its recursive nature, since the evaluation of $p_{i+1}(\boldsymbol{q})$ requires the calculation of $p_i(\boldsymbol{q})$ due to equation (5) which allows for an efficient calculation. The weights λ_i in the kernel k_{pose} control the importance of each position vector p_i. In the beginning we set all weights to 1 but discuss later how to improve imitation by changing these weights. After defining our pose kernel we are ready to predict the corresponding output features. Since we have to command our robot with joint values we have to reconstruct the final joint values from the predicted features. Thus we have to solve a pre-image problem. However, there is a nontrivial issue which is that we cannot provide an arbitrary pre-image vector $\boldsymbol{q}^*$ to our robot since we are constrained by the current state $\boldsymbol{q}^0$. In the next section we discuss how to construct pre-images which are close to the current state of the robot, but converge to final pose value.

6.2.2 Interpolation using constrained pre-images

In this section we discuss how to calculate the pre-image q^* given its estimated features Ψ under the constraint that the robot currently is in state q^0. [2] This constrained pre-image problem can be formulated as

$$q^* \quad = \quad \arg\min_{q \in \mathbb{R}^7} ||\boldsymbol{V}^\top \phi_{pose}(q) - \Psi||^2 + \lambda ||q - q_0||^2, \tag{7}$$

where ϕ_{pose} denotes the feature mapping corresponding to the pose kernel given in the previous section. Since Ψ is given by the predicted coordinates in some orthogonal basis $\boldsymbol{V}$ (see chapter 3) all quantities in (7) are computable. Note that the second term is not in the feature space since we really want to generate a pre-image which deviates as small as possible from the current state q_0 of the robot. Solving the pre-image problem (7) repeatedly by updating q_0 with q^* allows us to interpolate from q_0 to an optimal pre-image with intermediate joint values. Unfortunately, this requires that we use a specially tailored optimization approach such that the pre-image algorithms developed in chapter 3 cannot be used.

Alternatively, by omitting the second term in (7) we would have altered the problem back to the standard pre-image problem in chapter 3 which makes the developed techniques applicable. The simplest approach to create intermediate joint values is to use

[2] We thank Bernhard Schölkopf for this suggestion.

the unconstrained pre-image q^* as the end point for linear interpolation. This, however would conflict with our goal of imitating human movement. As an example, consider the first three features of training data in D_N plotted in figure 6.5 obtained by projecting all output joint vectors $\{y_i\}_{i=1}^N$ to the kernel PCA basis using the pose kernel. One can see that given two arbitrary joint vectors a linear interpolation yields a path in feature space that passes through low density regions (see figure 6.5-a) meaning that the robot moves in a manner which is probably not human-like. To ensure that the path stays in high density regions we propose to *correct* the linear interpolation path. This can be done by projecting the path into the feature space and looking for the point where the feature space distance to the nearest neighbor in D_N is maximal. The corresponding joint vector of the found nearest neighbor is then used as an intermediate pass point for the linear interpolation. This procedure is repeated until the maximal distance of a linear interpolation deviates not more than some given threshold. Since this path generation approach requires to calculate feature coordinates repeatedly it is obviously a performance bottleneck in the pre-image calculation. Therefore, we have to apply techniques from 4 to reduce the computational load in calculating kernel PCA features resulting in faster pre-image computation. We will discuss this later.

We now have all the necessary techniques to estimate output pose features and reconstruct the necessary output joint vector with the additional necessary intermediate points to steer the robot to the pre-image joint vector.

6.3 Experiments

In this section we want to investigate the generalization performance of our imitation system. Currently, the system is trained to follow a single actor (namely the author of this thesis). Nevertheless, we will test whether the system is capable to generalize to different actors of different skin color, body size and postures. We will start by describing the difficulties we faced in collecting the training data, an issue normally not discussed in machine learning. Then, we discuss the details of the parameterization and pre-processing we have used for KDE training. In particular, we will see the advantage of combining MRS and kPCA for KDE. Furthermore, we investigate whether the features are the right choice, i.e. if they fulfill the requirements mentioned in section 6.1.1. Since it is hard to quantify the imitation performance using an objective criteria, we investigate the imitated robot trajectories qualitatively. Let us start with the description of the data collection process.

Since KDE is a supervised approach we have to create a set of examples $D_N = \{x_i, y\}_{i=1}^N$ with human and robot posture information first. For the resulting N robot postures $\{y_i\}_{i=1}^N \in \mathbb{R}^7$ a human actor *imitated* the robot posture to obtain the required inputs x_i for each robot posture y_i. In this manner, we constructed 50 pairs of human-robot postures some of which are shown in figure 6.6. This dataset was used to learn the KDE mapping between human and robot postures which we are going to discuss next.

6.3.1 Training KDE

For KDE training we have to perform a kPCA both for the input and for output data. Afterwards we use the obtained principal components to extract non-linear features which are used to train the mapping T_F between feature spaces. To this end we will use the MRS technique from chapter 2 which will turn out to be very convenient if used in combination with kPCA. We finally apply the reduced set selection techniques from chapter 4 to speed

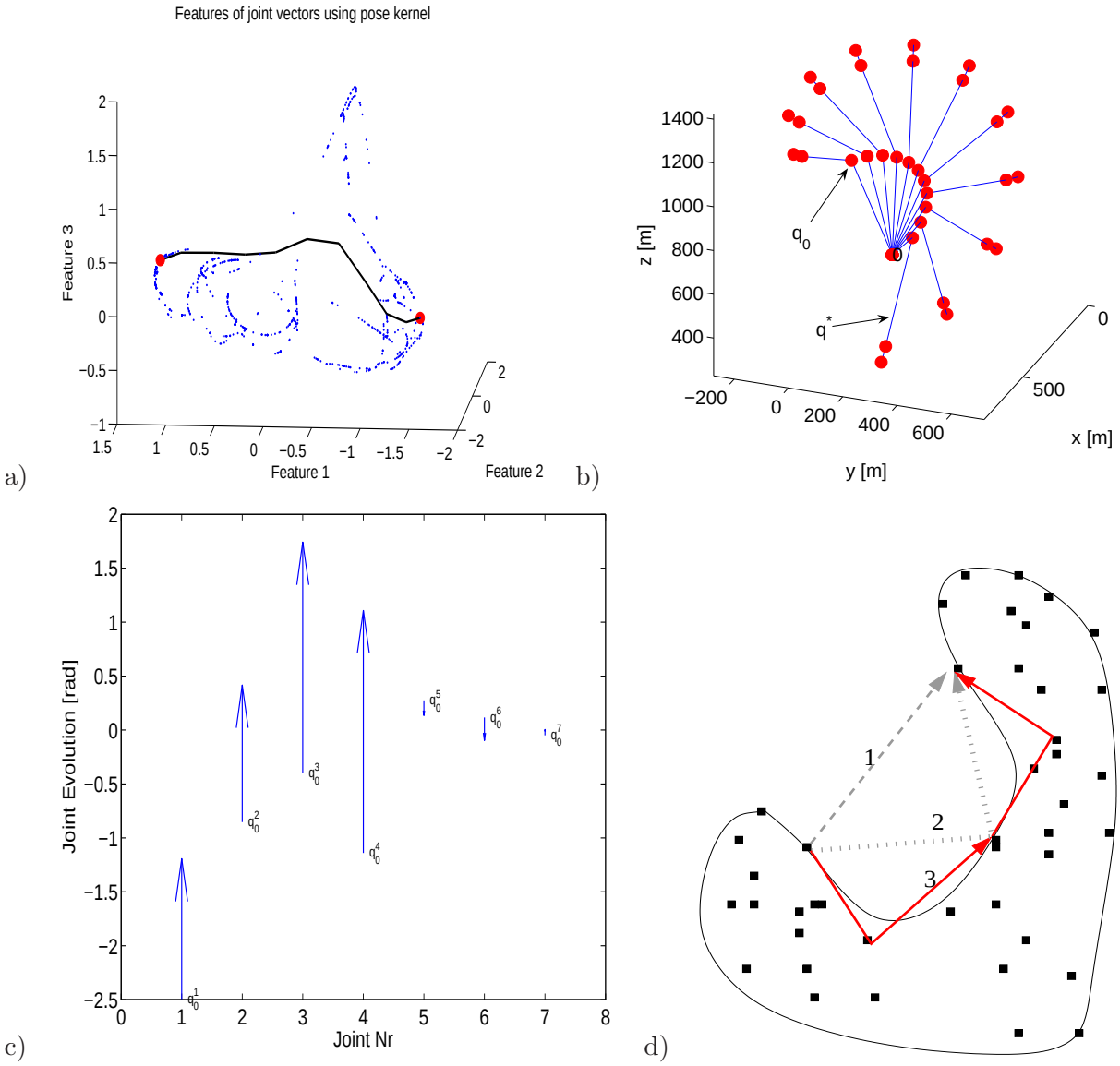


Figure 6.5: First three kPCA features of joint vectors during a linear interpolation of joints from q_0 to q^*. a) A linear interpolation of joints leads to a non-linear path in feature space. b) The robot postures during interpolation. c) The linear joint trajectories. d) Intermediate points are automatically generated if the linear interpolation between current state and found pre-image is too far from patterns in training set (path 1). This process must be repeated if necessary if the path resulting from new points is still too far (see path 2 and 3).

up the KDE map.

Kernel PCA on the output space. After obtaining output postures, we perform a kPCA using the weighted pose kernel k_{pose} introduced in (6). Since the used robot has the first two joint axis at the base (the shoulder), we can set the weights of the first two coefficients in the pose kernel to zero. This is possible since the Cartesian position of these joints cannot change due the kinematic of the used robot. Note that this does not imply that we can ignore the two joint values since they are needed for computation of all other

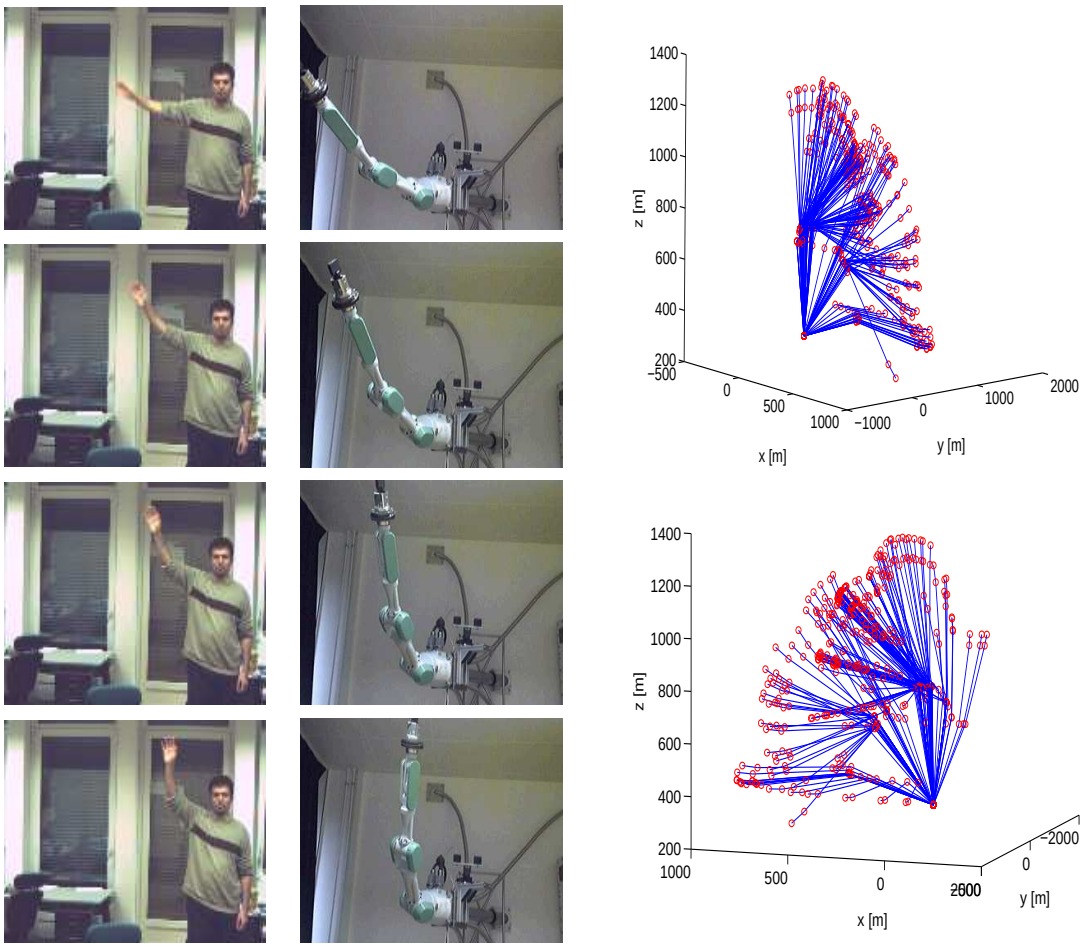


Figure 6.6: Example input output pairs of the training data. First column: Observed image. Second Column: Corresponding robot posture. Note that the algorithm does not use the images, but only the joint values. Third Column: Output training data set shown from two different camera azimuth values.

succeeding joint positions.[3] All remaining coefficients are set to one. Using this kernel we perform kPCA. The cumulative variance obtained from the eigenvalues is shown in figure 6.7. One can see that 95% of the variance is captured by the first 13 principal components which allows us to ignore the remaining ones. Let us denote the principal components found by kPCA by $\boldsymbol{V}_1 = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_{13}]$. Note that $\boldsymbol{v}_i$ is given as an expansion of training patterns, i.e.,

$$\boldsymbol{v}_i = s_i \sum_{j=1}^{50} e_i^j \phi_{pose}(y_j) = [\phi_{pose}(y_1), \ldots, \phi_{pose}(y_{50})]\, s_i e^i,$$

where $e^i \in \mathbb{R}^{50}$ is the coefficient vector obtained by the kPCA algorithm and s_i is a normalization constant such that $\boldsymbol{v}_i^\top \boldsymbol{v}_i = 1$, see A1.3. We will write all coefficients in one matrix E_v given as $E_v = [e^1, \ldots, e^{13}] \in \mathbb{R}^{50\times 13}$. Therefore the basis $\boldsymbol{V}_1$ is given as $\boldsymbol{V}_1 = [\phi_{pose}(y_1), \ldots, \phi_{pose}(y_{50})]\, SE_v$ and S_v is a diagonal matrix.

[3]The used kinematic map for our robot, the MITSUBISHI PA-10 is given explicitly in appendix X.1.

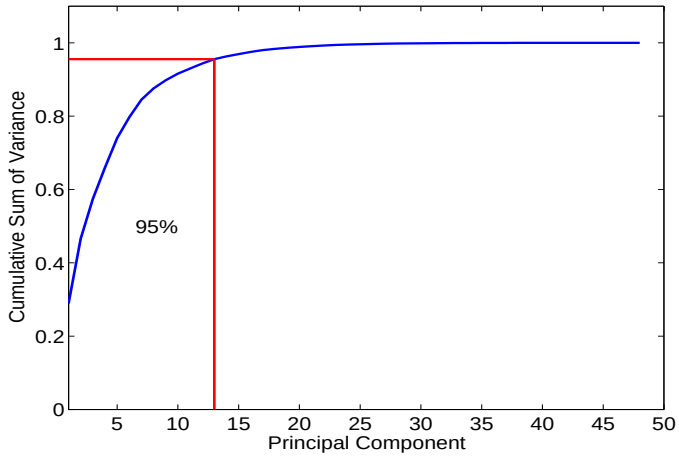


Figure 6.7: The spectrum of the feature space variance on the training data using the pose kernel. The first 13 components describe 95% of the observed variance.

Kernel PCA for the input space. Let us now estimate an orthogonal basis $\boldsymbol{W}_1$ in the feature space $\mathcal{F}_k$ of the input data. By carefully selecting the output data set we restrict ourselves to a small space of *valid* robot postures. Since both output and input bases are so far independent of each other we can use an arbitrary amount of data points to find $\boldsymbol{W}_1$. For the input case we will use more patterns besides the ones in D_{50} that are not necessarily assigned to a valid robot posture. This is necessary since we have to construct a finite codebook for the shape and color kernel first. To this end, we record 500 frames of an actor performing arm movements and calculate for all 500 frames the edge maps and shape contexts after resizing the image to 100×100. For shape context calculation we choose $d_k = e^{\frac{3k}{10}} - 1$ to obtain the distance bins and $\delta\phi = \frac{\pi}{3}$ as angular bin. Before calculating a shape descriptor containing all shape contexts we first smooth the image using a Gaussian convolution filter of size 5×5, where we use $\sigma = 3$ as variance parameter. From our 500 edge maps we retrieve 500 shape descriptors which consist of more than 20 shape contexts each, leading to a total number of 12475 shape contexts. We randomly choose 2000 shape contexts and perform vector quantization with the distance measure defined in (3). This results in 80 shape contexts which we use then use as codebook vectors. Thus if we obtain a shape descriptor $S = \{(e_i, H_i)\}_{i=1}^m$ of arbitrary size m we will calculate the resulting histogram by assigning each shape context $s_i \in S$ to one of the 80 bins. This is done by calculating the nearest neighbor of s_i given the 80 codebook vectors. For skin color estimation we manually select skin pixels from these 500 frames and use all selected color values to estimate the mean and the covariance of a single Gaussian distribution function. Overall we have 354 pixels for the estimation of color. This allows us now to calculate a skin map for each image and to calculate the row and column histograms h_x, h_y per image. We use 10 bins for h_x and h_y each.

Finally, we calculate the kPCA bases $\boldsymbol{W}_1$ in feature space via the color and shape kernel k_{s-c} introduced in (1). Let us denote the feature map associated to the this kernel by ϕ_{s-c}. We keep 109 principal components since they capture 95% of the variance (see figure 6.8). Analog as for the output bases $\boldsymbol{V}_1$ we yield for $\boldsymbol{W}_1 = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_{109}]$ expansions of training patterns

$$\boldsymbol{w}_i = s_i \sum_{j=1}^{500} e_i^j \phi_{s-c}(x_j) = [\phi_{s-c}(x_1), \ldots, \phi_{s-c}(x_{500})]\, s_i e^i,$$

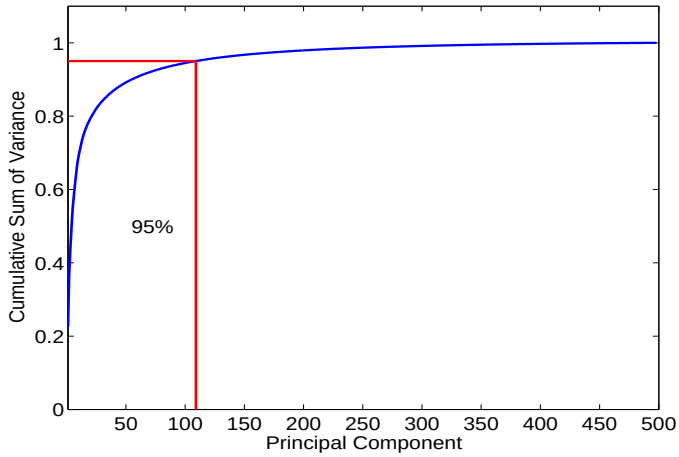


Figure 6.8: The spectrum of the feature space variance on the training data using the color and shape kernel. The first 109 components describe 95% of the observed variance.

where $e^i \in \mathbb{R}^{500}, s_i \in \mathbb{R}$ are the necessary coefficient vector and scaling coefficient. As for the output case we collect all coefficients in one matrix E_w given as $E_w = [e^1, \ldots, e^{109}] \in \mathbb{R}^{500\times 109}$.

Since we can use the input and output bases $\boldsymbol{V}_1$ and $\boldsymbol{W}_1$ for feature extraction we extract features of all data pairs in D_{50}. Thus we extract a 109-dimensional description of the input image and a 13-dimensional description of the robot posture. Let us denote the coordinate of a data point x in feature space by $\hat{x}$. Thus we obtain the new dataset $\hat{D}_{50} = \{\hat{x}_i, \hat{y}_i\}_{i=1}^{50}$ where $\hat{x}_i = \boldsymbol{W}_1^\top \phi_{s-c}(x_i)$ and $\hat{y}_i = \boldsymbol{V}_1^\top \phi_{pose}(y_i)$. Using $\hat{D}_{50}$ we now have to learn the KDE mapping between the coordinates of images and postures. For this mapping, we use MRS.

Training MRS and speeding up Since we have only 50 data points to estimate a linear mapping from 109 to 13 dimensions we have to check whether the data points are possibly orthogonal in the feature space of the inputs. In this case, the application of a rank-constrained regression technique such as MRS would not make sense. Fortunately, as one can see in figure 6.9, this is not the case. It is reasonable to assume that all data live in a 13-dimensional subspace. To find a reasonable dimension of the subspace and thus a rank parameter for MRS, we performed 5 fold cross validation. We tested the rank values [5,...,13] with rank 7 being the best choice. Note that, it is not necessary to test larger rank values than 13 since this is the number of principal components we kept in the output feature space. Note that the MRS mapping yields us an explicit SVD presentation of the predictor T_F. Let us denote this SVD representation as

$$T_F = V_2 D W_2^\top,$$

where in our case $W_2 \in \mathbb{R}^{109\times 7}$, $V_2 \in \mathbb{R}^{13\times 7}$ are orthogonal matrices of rank r and $D \in \mathbb{R}^{7\times 7}$ is a 7 by 7 diagonal matrix. We assessed the performance of MRS by comparison to PLS. We perform 5 fold cross validation where at each fold we train MRS with rank constrained equal to 7 and PLS with 7 iterations. We obtained a squared crossvalidation error of 14.14 ± 1.5 for PLS and 13.8 ± 1.4 for MRS. This indicates that our regression method performed better than a standard reduced rank technique.

The benefit of using kPCA and MRS together is that we can multiply W_2 by the coefficient matrix E_w to obtain thew expansion matrix $\bar{E}_w = E_w W_2$. This can be considered as

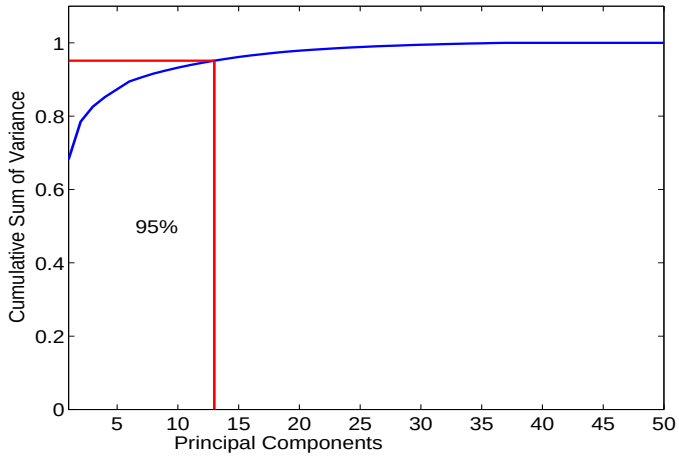


Figure 6.9: The cumulative sum of variances of the training data in feature space. The data lives in a subspace of the 109-dimensional feature space.

a rotation in feature space $\boldsymbol{W}_1 \rightarrow \boldsymbol{W}_2$ by applying the orthogonal transformation W_2 to the coefficient matrix E_w (by definition of kPCA E_w is also orthogonal). Thus we obtain the new orthogonal basis $\boldsymbol{W}_2$

$$\boldsymbol{W}_2 = \boldsymbol{X}\bar{S}\underbrace{E_w W_2}_{orthogonal} = \boldsymbol{X}\bar{S}\bar{E}_w.$$

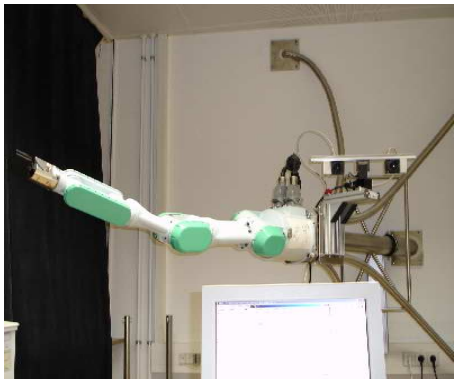
Here $\bar{S}$ denotes a diagonal matrix which ensures that the columns of $\boldsymbol{W}_2$ have unit norm. Fortunately $\bar{E}_w$ has a lower rank than E_w. This implies that we will need fewer principal directions, such that input feature extraction can be accelerated. Indeed in our case we obtained a reduction from 109×500 operations necessary for feature extraction down to 7×500 operations. This is already a substantial reduction, but further speed up is possible by using reduced set selection. The application of multi-hyperplane matching pursuit yielded a final modified kernel PCA expansion which uses only 167 input patterns from 500 original ones.

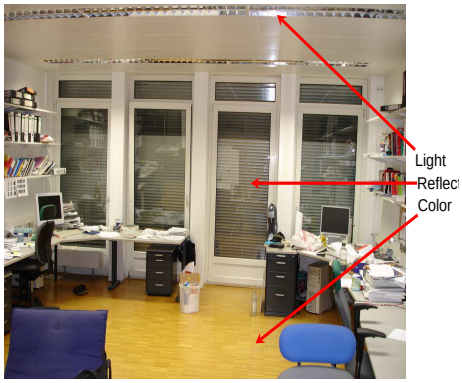
Pre-images After prediction with MRS we have to reconstruct the real joint vector from its estimated coordinates. To achieve this task, we use a one-nearest-neighbor approach to learn the pre-image map. In our case a learning approach based on nearest-neighbor is well suited since we want to choose a joint configuration from a (possibly large) set of a-priori set robot postures. The found pre-images are then used for trajectory generation as described in section 6.2.2.

6.3.2 Are the features the right choice?

Since our robot and its camera is mounted on a wall (see figure 6.10-a), the background of the images will be almost static. However, it turned out that although we are working in an indoor scene, background clutter and varying influences are sources of error. To see this, consider the indoor scene shown in figure 6.10-b where the actors are observed. Since all actors are standing upright, the least critical issue is point C: The *skin-colored* floor. Unfortunately, both other issues might lead to problems. In particular, color turns out to be a fragile and sensitive clue. Currently our system forces us to re-calibrate the skin-color detector depending on the day time since the environmental lighting Therefore

the color values change during the day. Furthermore, we currently are not able to avoid wrong contours which caused by reflections in the window. For example consider the contour in figure 6.10-c where the actor rises his right arm. One can see that an additional "ghost-arm" arm appears on his head. Such erroneous contours appear due to reflections in the background and are an artifact by our naive background subtraction.

a) 

b) 

c) 

Figure 6.10: a) Our robot is mounted on a wall and we are operating in an indoor scene. Thus the assumption of a constant background is reasonable. b) Sources of problems for our assumed conditions. A: Lighting conditions change due to daylight, B: Curtains can move and day light might pass in leading to clutter. Furthermore, windows add reflections in the scene leading to false contours as can be seen in c). C: The color of the ground floor can appear as skin due the lighting.

Let us now investigate some invariance and stability properties of the used shape and color features. Although our features are constructed such that they are invariant under scale and translation it will turn out that in practice invariance properties are lost due to the experimental setup and the preprocessing. To this end, we perform the following experiment: we take snapshots of an actor showing two distinct postures at various positions allowing for motion only parallel to the camera. We would expect a translation invariant feature to yield feature vectors that do not change during translation. Thus if we plot the distance between two feature vectors we would expect a value close to zero. In figure 6.11 we show the translated postures and the resulting distance plots. One can see that indeed that there is a clear gap between both curves. However, there is some small variation in the path of features. Does this mean that our features are not translation invariant? The answer is yes and no. No, because our features are invariant by construction since position and scale does not enter. Unfortunately, complete invariance cannot maintained since we have to resize the image to 100 by 100. Thus we introduce small quantization artifacts although the image is smoothened by a Gaussian filter. On

the other hand, another source of instability of the features is that we do not have uniform lighting in the room. Thus small changes in the physical location might lead instabilities in the contour due to the inhomogeneous lighting.

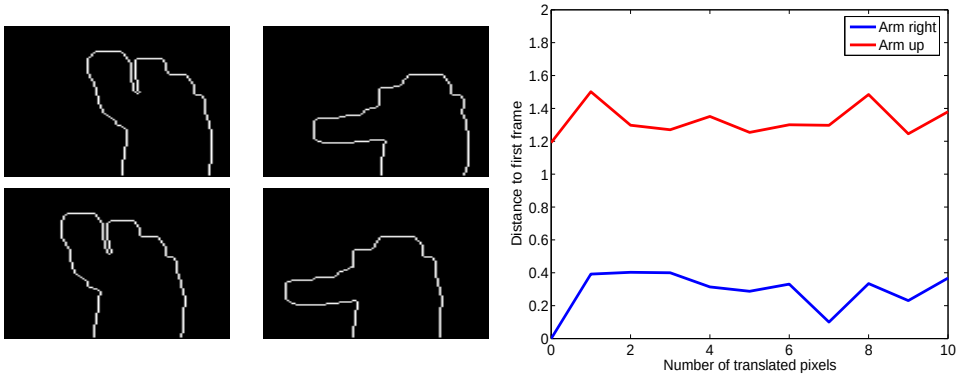


Figure 6.11: Source of variation in the feature vector. Although we are using theoretically translation-invariant features the real feature vectors do not fulfill this requirement entirely. This effect might be due to quantization artifacts and possible varying lighting conditions in the room.

After discussing these difficulties, the reader will have lowered his expectations sufficiently. Let us therefore advance and discuss how our system performs in reality.

Resulting Trajectory Paths

In the following we record simple movements from various actors and investigate the resulting robot joint trajectories. Ideally we expect trajectories which lead to a posture similar to the actors. We present successfully imitated postures and cases where the system failed. For these unsuccessful cases we discuss reasons for the failure and possible improvements.

Successful Cases

In the following we let an actor perform arm-up-down movements and record the synthesized trajectories of our system. We show in figure 6.12 the resulting joint trajectories for the first 6 joints while an actor rises his right arm. The obtained trajectories are for a sequence of 10 moves. As one can see, smooth trajectories are obtained. Furthermore the resulting trajectories lead to arm motion which resembles an arm-down to arm-up movement. In figure 6.12 we show the result of an arm waving sequence. An interesting fact is that during imitation the human adjusts his speed to the robot. For example, a complete arm rising movement took almost 10 seconds. However, this is still faster than the computationally expensive optimization procedure proposed in chapter 5.

Unsuccessful Cases

Let us now discuss some cases where the system was not able to find a reasonable robot posture. Consider the examples in figure 6.14. Since the shape and color feature vectors are quite distinct from anything we saw in the training data set, it is unlikely to expect a good generalization for these scenarios. Currently, the system reacts as it would have predicted with a zero feature vector since the the observed features are so different from

the training data and therefore the color histograms are zero. Thus the system does not respond in an uncontrolled manner if it is not able to generalize.

6.4 Conclusion

In this chapter we considered the problem of posture imitation as an estimation task and discussed a possible implementation. In particular, we have formulated the estimation task as: given a single image of a human actor what is the corresponding robot posture? We proposed to use KDE for the problem of learning the map between image and posture. Since KDE requires to define similarity measures for input and output we have proposed

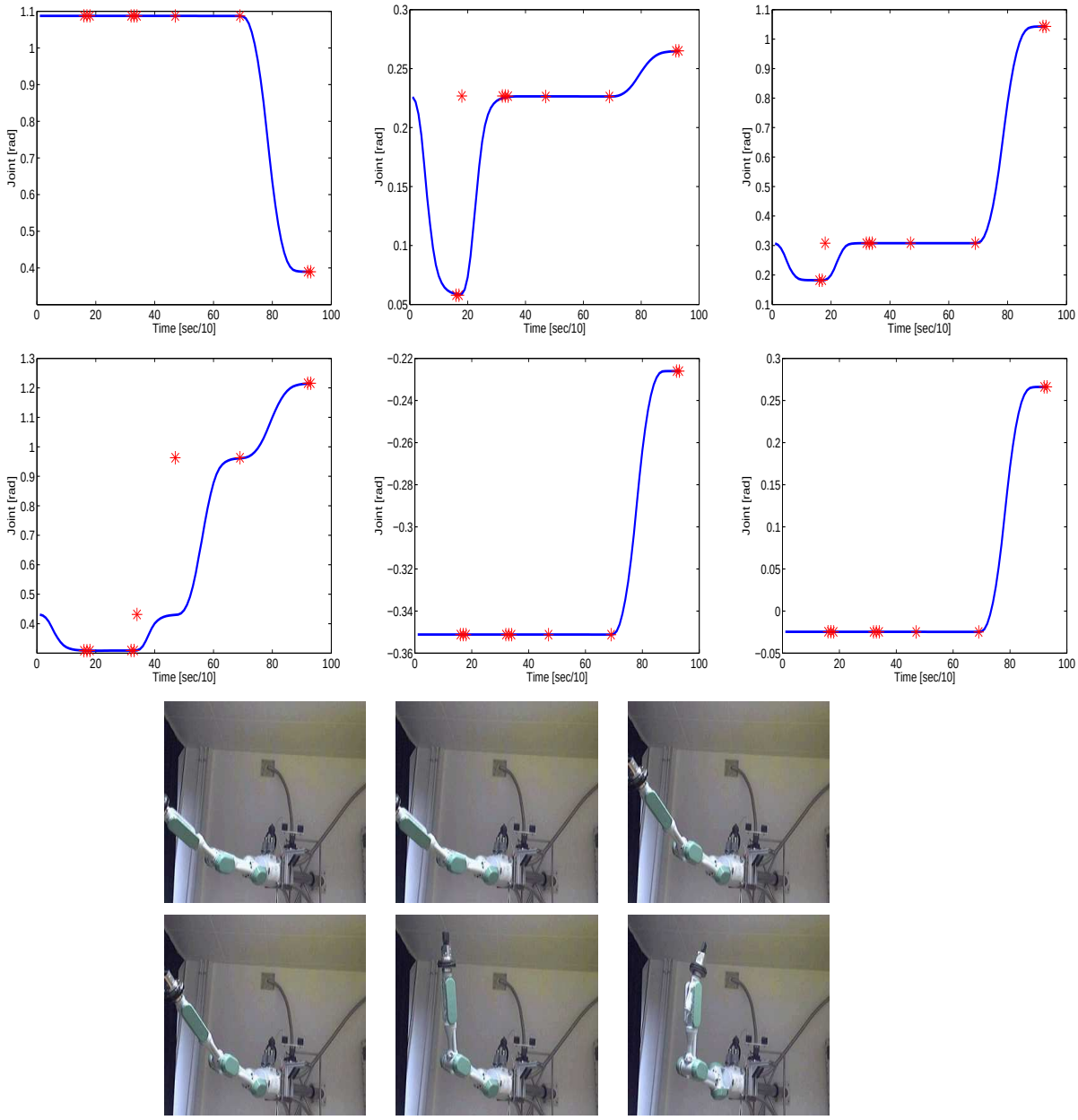


Figure 6.12: Joint trajectories while imitating an arm-down to arm-up movement.

special kernel functions which are suitable to measure similarity between robot postures. Therefore, we have investigated shape contexts and color as features which represent human posture in a single image. The final system is capable of imitating simple movements after a teach-in phase consisting of a human imitating a set of robot postures. In contrast to chapter 5 the learning-based system is able to imitate human posture almost online. Currently, the most unreliable part in the system is the skin color extraction. We have shown that the system is not able to generalize to a different actor with a different skin color if this actor was not the teacher. In future, we will investigate more robust color features and try to integrate information over time. Since KDE uses information only through kernel functions it is independent of the nature of the used features. Thus more

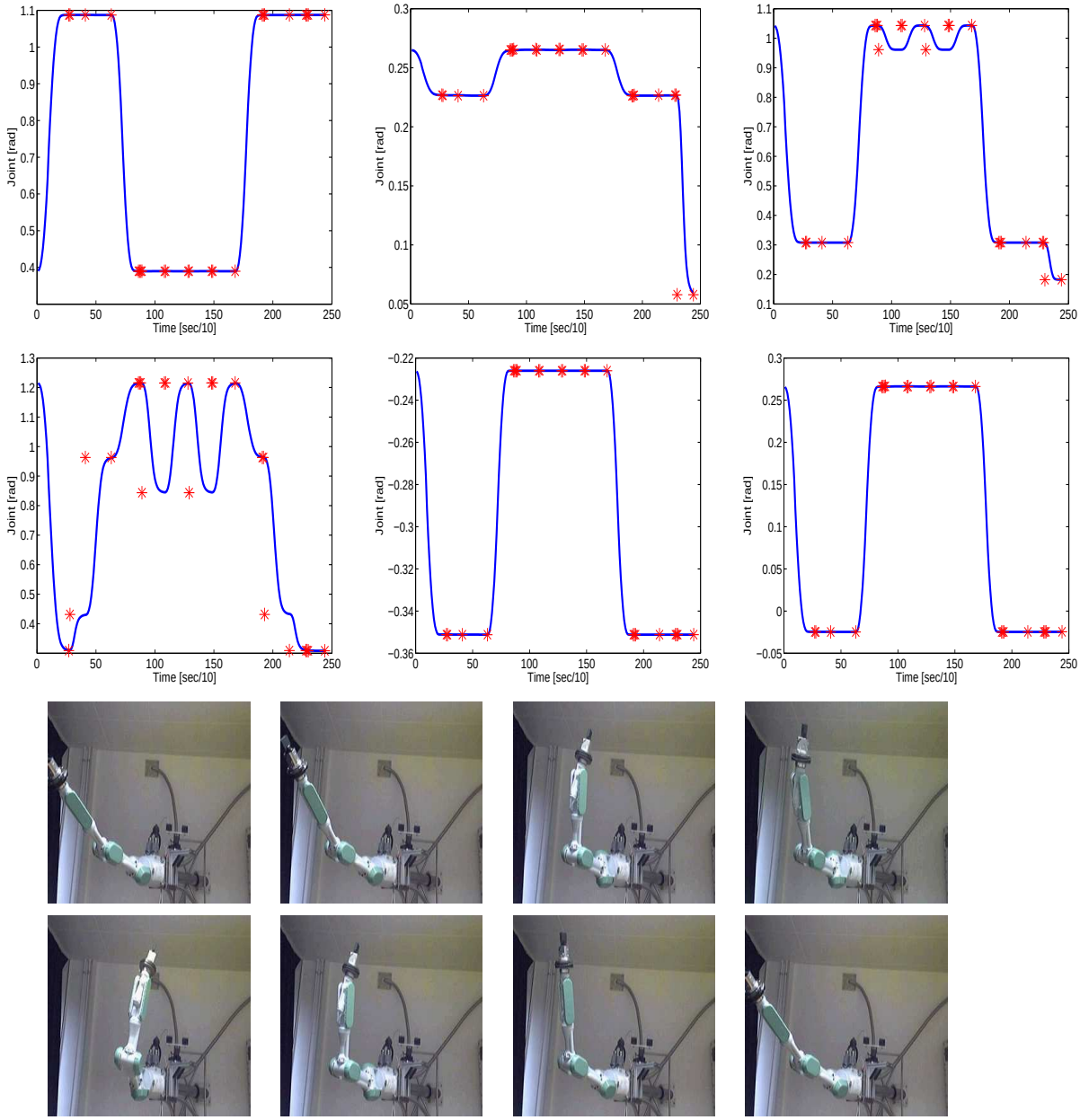


Figure 6.13: Joint trajectories and robot postures while imitating an arm-waving movement.

robust pose features can be considered and easily integrated into the current framework. This clearly demonstrates the strength of KDE: Feature encapsulation by kernel functions.

Figure 6.14: Examples of failure. Background clutter and unreliable color information leads to an erroneous reaction of the system. First column: Observed image. Second column: Observed edge map. The resulting color map was overall zero since the skin color was different from those during training. Thus no skin information was found. Third column: Estimated pose. As one can see the system does not generalize in this case.

Chapter 7

Summary

You are smarter than computer
– Vladimir Vapnik –

In this thesis we have explored various algorithmic extensions to the method of Kernel Depency Estimation(KDE) and its application to the problem of robot imitation.

Kernel Dependency Estimation is a novel technique which was designed to learn mappings between sets without making assumptions on the type of the involved input and output data. KDE uses kernel functions to encapsulate task-dependent information and reformulates the original problem as a high-dimensional regression problem in feature space. One of the contributions of this thesis is to propose a novel linear regression algorithm in chapter 2 that explores low-dimensional subspaces *during* learning. Learning is formulated as a single optimization problem while low-dimensional subspace exploration is ensured by a rank constraint. Problems with rank constraints can be solved by exploiting the geometrical structure – the Stiefel manifolds – implied by the rank constraint. For the case that the desired cost function is not differentiable we explore Markov Chain Monte Carlo (MCMC) techniques to obtain the maximum a posteriori predictor using again properties of the Stiefel manifold while performing MCMC.

In KDE, after estimating data-type independent feature space coordinates of a new output entity, one has to reverse the feature mapping by solving the so called pre-image problem. We propose various techniques how to approach this problem. In the case that the set of pre-images are smooth, we propose in chapter 3 a learning-based approach. This has the advantage that the complexity of finding the pre-image is mainly shifted to a training phase. Thus, the pre-image is merely found by an estimation procedure instead of a search. We have investigated another pre-image method for data-types that are of discrete nature such as graphs and strings. In particular we explored techniques based on the Cross-Entropy (CE) method and extended them to a maximum a posteriori estimate. This is the first pre-image technique for graps nd strings we are aware of which is in principle independent of the kernel function.

A bottleneck in the use of kernel methods, and in particular in KDE is that solutions to a learning task are always an expansion of kernel functions on training points. For example, kernel PCA always uses all training patterns to express its solutions. For realtime applications this is a serious drawback. Therefore, we investigated reduced set selection

techniques in chapter 4 which are able to find sparser subsets. In contrast to existing methods, we have proposed two novel reduced set selection techniques which are suitable to compress several expansions simultanously, as, e.g. obtained from kernel PCA.

In chapter 5 we have investigated our main application of interest: the problem of imitating human posture and its implementation on a seven degree of freedom robot arm. So far, this problem was not clearly defined in the literature. To this end, we proposed in chapter 5 a mathematical cost function where we have introduced a novel geometrical description of posture. The advantage of these geometrical description is that it is independent of the underlying kinematics. We used this cost function to formulate the problem of posture imitation as a non-linear optimization approach. Given a human arm trajectory we then found the corresponding robot trajectory as the solution to an optimization problem. We also showed empirically that exact imitation of pose while performing a pre-defined task leads to sub-optimal robot trajectories.

Finally, in chapter 6 we relaxed the requirement of knowing the exact 3D-trajectory of the human. We considered the case when a single input image is given without extra information about the pose of the human actor. In this setting, an optimization approach is not possible anymore since the necessary quantities – the posture – is not directly observable. Therefore, we have investigated the use of image features, i.e. shape and color, to describe posture in an image. The corresponding robot posture was inferred using KDE combined with our rank-constrained regression method. Since KDE requires the specification of kernel functions we have introduced a novel kernel function which can be used to measure similarity between robot postures. This was not possible with existing kernel functions. We implemented the learning-based imitation system on a real robot system and demonstrated that KDE is a suitable learning framework for problems in robotics.

Appendix

X.1 The kinematical equations of the Mitsubishi PA-10 robot

The forward kinematical map $F_k(\boldsymbol{q})$ of the Mitsubishi PA-10 robot can be written as a product of k transformation matrices

$$F_k(\boldsymbol{q}) = A_1(q_1)A_2(q_2)\cdots A_k(q_k).$$

The matrices are given as follows and are taken from [97]:

$$A_1(q_1) = \begin{bmatrix} \cos(q_1) & -\sin(q_1) & 0 & 0 \\ \sin(q_1) & \cos(q_1) & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_2(q_2) = \begin{bmatrix} \cos(q_2) & -\sin(q_2) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin(q_2) & -\cos(q_2) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3(q_3) = \begin{bmatrix} \cos(q_3) & -\sin(q_3) & 0 & 0 \\ 0 & 0 & -1 & -450 \\ \sin(q_3) & \cos(q_3) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_4(q_4) = \begin{bmatrix} \cos(q_4) & -\sin(q_4) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin(q_4) & -\cos(q_4) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_5(q_5) = \begin{bmatrix} \cos(q_5) & -\sin(q_5) & 0 & 0 \\ 0 & 0 & -1 & -500 \\ \sin(q_5) & \cos(q_5) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_6(q_6) = \begin{bmatrix} \cos(q_6) & -\sin(q_6) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\sin(q_6) & -\cos(q_6) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_7(q_7) = \begin{bmatrix} \cos(q_7) & -\sin(q_7) & 0 & 0 \\ 0 & 0 & -1 & -80 \\ \sin(q_7) & \cos(q_7) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Bibliography

[1] Ankur Agarwal and Bill Triggs. Learning to track 3d human motion from silhouettes. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 2, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-828-5. doi: http://doi.acm.org/10.1145/1015330.1015343.

[2] Gökhan H. Bakır, Léon Bottou, and Jason Weston. Breaking svm complexity with cross-training. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 81–88. MIT Press, Cambridge, MA, 2005.

[3] Serge Belongie, Jitendra Malik, and Jan Puzicha. Shape context: A new descriptor for shape matching and object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(24):509–522, 2002.

[4] K. P. Bennett and M. J. Embrechts. An optimization perspective on partial least squares regression. In S. Basu C. Micchelli J. Vandewalle J.A.K. Suykens, G. Horvath, editor, *Advances in Learning Theory: Methods, Models and Applications*, volume 190, pages 227–250. IOS Press, 2003.

[5] A. F. Bobick and J. Davis. An appearance-based representation of action. *Pattern Recognition*, 1:307–312, August 1996.

[6] A. W. Bojanczyk and A. Lutoborski. The Procrustes problem for orthogonal Stiefel matrices. *SIAM Journal on Scientific Computing*, 21(4):1291–1304, 2000.

[7] Matthew Brand and Aaron Hertzmann. Style machines. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 183–192. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[8] Matthieu Bray, Esther Koller-Meier, Nicol N. Schraudolph, and Luc Van Gool. Stochastic meta-descent for tracking articulated structures. In *Conference on Computer Vision and Pattern Recognition*, Washington D.C., 2004.

[9] C. Bregler and J. Malik. Tracking people with twists and exponential maps. In *IEEE Computer Vision and Pattern Recognition*, pages 8–15, 1998.

[10] C. Bregler, L. Loeb, E. Chuang, and H. Deshpande. Turning to the masters: Motion capturing cartoons. *ACM Transactions on Graphics*, 21(3):399–407, 2002.

[11] A. Bruderlin and L. Williams. Motion signal processing. In *SIGGRAPH*, pages 97–104, 1995.

[12] C. J. C. Burges. Simplified support vector decision rules. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning*, pages 71–77, San Mateo, CA, 1996. Morgan Kaufmann.

[13] C. J. C. Burges and B. Schölkopf. Improving the accuracy and speed of support vector learning machines. In M. Mozer, M. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems 9*, pages 375–381, Cambridge, MA, 1997. MIT Press.

[14] T. Caelli, A. McCabe, and G. Binsted. On learning the shape of complex actions. In *International Workshop on Visual Form*, pages 24–39, 2001.

[15] Yasuko Chikuse. Concentrated matrix langevin distributions. *Journal of Multivariate Analysis*, 85:375394, 2003.

[16] M. Crampin and F. A. E. Pirani. *Applicable Differential Geometry.* Cambridge University Press, Cambridge, U. K., 1986.

[17] Gabriela Csurka, Cedric Bray, Chris Dance, and Lixin Fan. Visual categorization with bags of keypoints. In *Proceedings of the 8th European Conference on Computer Vision - ECCV*, 2004.

[18] S. de Jong. Simpls: An alternative approach to partial least squares regression. *Chemometrics and Intelligent Laboratory Systems*, 18:251–263, 1993.

[19] A.K. Debnath, R.L. Lopez de Compadre, G. Debnath, A.J. Shustermann, and C. Hansch. Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. *J Med Chem*, 34:786–797, 1991.

[20] D. di Ruscio. A weighted view on the partial least squares algorithm. *Automatica*, 36:831–850, 2000.

[21] T. Downs, K. E. Gates, and A. Masters. Exact simplification of support vector solutions. *Journal of Machine Learning Research*, 2:293–297, December 2001.

[22] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification.* Wiley, New York, second edition, 2001.

[23] A. Edelman, T. A. Arias, and S. T. Smith. The geometry of algorithms with orthogonality constraints. *SIAM Journal on Matrix Analysis and Applications*, 20(2): 303–353, 1999.

[24] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, New York, 1989.

[25] A. Fod, M. J. Mataric, and O. C. Jenkins. Automated derivation of primitives for movement classification. *Autonomous Robots*, 12(1):39–54, 2002.

[26] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns.* Addison-Wesley Professional, 1995.

[27] D.M. Gavrila and L. Davis. 3d model-based tracking of humans in action: A multi-view approach. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition*, pages 73–80, 1996.

[28] C. Gentile. A new approximate maximal margin classification algorithm. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems 13*, pages 500–506. MIT Press, 2001.

[29] Alan Genz. Methods for generating random orthogonal matrices. In H. Niederreiter and J. Spanier, editors, *Monte Carlo and Quasi-Monte Carlo Methods*, pages 199–213. Berlin, 1999.

[30] J. Giebel, D.M. Gavril, and C. Schnörr. A bayesian framework for multi-cue 3d object tracking. In T. Pajdla and J. Matas, editors, *Proceedings of the European Conference on Computer Vision (ECCV)*, 2004.

[31] M. A. Giese and T. Poggio. Synthesis and recognition of biological motion pattern based on linear superposition of prototypical motion sequences. In *Proceedings of IEEE MVIEW 99 Symposium at CVPR, Fort Collins*, pages 73–80, 1999.

[32] M.A. Giese and T. Poggio. Morphable models for the analysis and synthesis of complex motion patterns. *International Journal of Computer Vision*, 38(1):59–73, 2000.

[33] M.A. Giese, B. Knappmeyer, and H.H. Bülthoff. Automatic synthesis of sequences of human movements by linear combination of learned example patterns. In *Workshop on Biologically Motivated Computer Vision*, pages 538–547, 2002.

[34] G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, Baltimore, MD, 3rd edition, 1996.

[35] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer Verlag, New York, 2001.

[36] M. Hein and O. Bousquet. Hilbertian metrics and positive definite kernels on probability measures. In *Proceedings of AISTATS 2005*, 2005.

[37] I. Helland. On the structure of partial least squares regression. *Commun. Statist. Simula.*, 17(2):581–607, 1988.

[38] A.E. Hoerl and R.W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(3):55–67, 1970.

[39] B. Hofmann. *Mathematik inverser Probleme*. B.G. Teubner Stuttgart-Leipzig, Leipzig, 1999.

[40] W. Ilg and M.A. Giese. Modeling of movement sequences based on hierarchical spatial-temporal correspondence of movement primitives. In *Workshop on Biologically Motivated Computer Vision*, pages 528–537, 2002.

[41] W. Ilg and M.A. Giese. Estimation of skill level in sports based on hierarchical spatio-temporal correspondences. 2003. submitted.

[42] C. J. Isham. *Modern Differential Geometry for Physicists*, volume 61 of *World Scientific Lecture Notes in Physics*. World Scientific, Singapore, second edition, 1999.

[43] O.C. Jenkins and M. J. Mataric. Deriving action and behavior primitives from human motion data. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2551–2556, 2002.

[44] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, New York, New York, 1986.

[45] Michael J. Jones and James M. Rehg. Statistical color models with application to skin detection. *International Journal of Computer Vision*, 46(1):81 – 96, 2002.

[46] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In T. Faucett and N. Mishra, editors, *Proceedings of the 20th International Conference on Machine Learning*, pages 321–328, Menlo Park, CA, AAAI Press, 2003.

[47] M. Kass, A.P. Witkin, and D. Terzopoulos. Snakes: Active contour models. *IJCV*, 1(4):321–331, January 1988.

[48] Wolf Kienzle, Gökhan H. Bakır, Matthias O. Franz, and Bernhard Schölkopf. Face detection — efficient and rank deficient. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 673–680. MIT Press, Cambridge, MA, 2005.

[49] K.I. Kim, M.O. Franz, and B. Schölkopf. Iterative kernel principal component analysis for image modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2003.

[50] G. S. Kimeldorf and G. Wahba. Some results on Tchebycheffian spline functions. *Journal of Mathematical Analysis and Applications*, 33:82–95, 1971.

[51] C. Klein and B. Baho. Dexterity measures for the design and control of kinematically redundant manipulators. *International Journal of Robotic Research*, 6(2):72–83, 1987.

[52] Donald E. Knuth. *The Art of Computer Programming*. Addison-Wesley, 1998.

[53] R. Kondor and T. Jebara. A kernel between sets of vectors. In *Proceedings of the ICML*, 2003.

[54] J.T. Kwok and I.W. Tsang. The pre-image problem in kernel methods. *IEEE Transactions on Neural Networks*, 15(6):1517–1525, 2004.

[55] F. Lindgren, P. Geladi, and S. Wold. The kernel algorithm for pls. *Journal of Chemometrics*, 7:45–60, 1993.

[56] J. S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer Series in Statistics. Springer Verlag, New York, 2001.

[57] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. CUP, 2003.

[58] O. Mangasarian. *Nonlinear Programming*. SIAM, Philadelphia, 1994.

[59] M. J. Mataric. Visuo-motor primitives as a basis for learning by imitation : Linking perception to action and biology to robotics. In K. Dautenhahn and C. Nehaniv, editors, *Imitation in Animals and Artifacts*, pages 392–422. MIT Press, 2002.

[60] C. A. Micchelli. Interpolation of scattered data: distance matrices and conditionally positive definite functions. *Constructive Approximation*, 2:11–22, 1986.

[61] S. Mika, B. Schölkopf, A.J. Smola, K.-R. Müller, M. Scholz, and G. Rätsch. Kernel pca and de-noising in feature spaces. volume 11 of *Advances in Neural Information Processing Systems*, pages 536–542. MIT PRESS, 1999.

[62] Anurag Mittal, Liang Zhao, and Larry S. Davis. Human body pose estimation using silhouette shape analysis. In *IEEE Conference on Advanced Video and Signal Based Surveillance*, pages 263– 270, 2003.

[63] H. Miyamoto and M. Kawato. A tennis serve and upswing learning based on bi-directional theory. *Neural Networks*, 11:1331–1344, 1998.

[64] Greg Mori and Jitendra Malik. Estimating human body configurations using shape context matching. In *ECCV (3)*, pages 666–680, 2002. URL `citeseer.ist.psu.edu/mori02estimating.html`.

[65] T. Mori and K. Uehara. Extraction of primitive motion and discovery of association rules from motion data. In *Proceedings of the IEEE International Workshop on Robot and Human Interactive Communication*, pages 200–206, 2001.

[66] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, MA, 1995.

[67] R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993. URL `citeseer.ist.psu.edu/neal93probabilistic.html`.

[68] Maria Petrou and Panagiota Bosdogianni. *Image Processing, The Fundamentals*. John Wiley and Sons, 1999.

[69] N. S. Pollard, J. Hodgins, M. J. Riley, and C. Atkeson. Adapting human motion for the control of a humanoid robot. In *IEEE International Conference on Robotics and Automation*, 2002.

[70] G. Rätsch, T. Onoda, and K.-R. Müller. Soft margins for AdaBoost. *Machine Learning*, 42(3):287–320, 2001. Also: NeuroCOLT Technical Report 1998-021.

[71] C. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–40, 1998.

[72] Roman Rosipal and Leonard J. Trejo. Kernel partial least squares regression in reproducing kernel hilbert space. *JMLR Special Issue on Kernel Methods*, Special Issues, 2001.

[73] H. A. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(1):23–38, 1998.

[74] Reuven Rubinstein and Dirk Kroese. *The Cross-Entropy Method: A Unified Approach To Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning*. Springer, New York, USA, 2004.

[75] J. Salisbury and J. Craig. Articulated hands: Force control and kinematic issues. *Int. J.of Robotic Research*, 1982.

[76] Ali H. Sayed. *Fundamentals of Adaptive Filtering*. Wiley-IEEE Press, 2003.

[77] S. Schaal. Is imitation learning a route to humanoid robots. *Trends in Cognitive Science*, 3:233–242, 1999.

[78] B. Schölkopf and A. J. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[79] B. Schölkopf, S. Mika, C. Burges, P. Knirsch, K.-R. Müller, G. Rätsch, and A. J. Smola. Input space vs. feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, 10(5):1000–1017, 1999.

[80] B. Schölkopf, A. J. Smola, and K.-R. Müller. Kernel principal component analysis. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*, pages 327–352. MIT Press, Cambridge, MA, 1999.

[81] B. Schölkopf, K. Tsuda, and J.P. Vert. *Kernel Methods in Computational Biology*. MIT Press, 2004.

[82] G. Schreiber, C. Ott, and G. Hirzinger. Interactive redundant robotics: Control of the inverted pendulum with nullspace motion. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2001.

[83] L. Sciavicco and B. Siciliano. *Modelling and Control of Robot Manipulators*. McGraw-Hill, New York, NY, 1996.

[84] L. Sciavicco and B. Siciliano. *Modeling and Control of Robot Manipulators*. 1996.

[85] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[86] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, 1994.

[87] I. Steinwart. On the generalization ability of support vector machines. *Journal of Machine Learning*, 2:67–93, December 2001.

[88] Bjarne Stroustrup. What is object-oriented programming? *IEEE Softw.*, 5(3):10–20, 1988. ISSN 0740-7459. doi: http://dx.doi.org/10.1109/52.2020.

[89] Robert Tempo, Giuseppe Calafiore, and Fabrizio Dabbene. *Randomized Algorithms for Analysis and Control of Uncertain Systems*. Springer-Verlag, 2004.

[90] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-Posed Problems*. V.H. Winston & Sons, John Wiley & Sons, Washington D.C., 1977. Translation editor Fritz John.

[91] M. Tipping and B. Schölkopf. A kernel approach for vector quantization with guaranteed distortion bounds. In T. Jaakkola and T. Richardson, editors, *Artificial Intelligence and Statistics*, pages 129–134, San Francisco, CA, 2001. Morgan Kaufmann.

[92] Silvio Turrini. Optimization in permutation spaces. Technical Report WRL Research Report 96/1, digital Western Research Laboratory, 1996.

[93] A. Ude, C. G. Atkenson, and M. Riley. Planning of joint trajectories for humanoid robots using b-spline wavelets. In *IEEE International Conference on Robotics and Automation*, 2000.

[94] M. Unuma, K. Anjyo, and R. Takeuchi. Fourier principles for emotion-based human figure animation. In *SIGGRAPH*, pages 91–96, 1995.

[95] Vladimir Vapnik. *Statistical Learning Theory*. John Wiley and Sons, New York, 1998.

[96] P. Vincent and Y. Bengio. Kernel matching pursuit. *Machine Learning*, 48:165–187, 2002.

[97] Jun Wang, Qingni Hu, and Danchi Jiang. A lagrangian network for kinematic control of redundant robot manipulators. *IEEE Transactions on Neural Networks*, 10(5), 1999.

[98] J. Weston, A. Elisseeff, and B. Schölkopf. Use of the ℓ_0-norm with linear models and kernel methods. Technical report, Biowulf Technologies, New York, 2001.

[99] Jason Weston, Olivier Chapelle, Andre Elisseeff, Bernhard Schölkopf, and Vladimir Vapnik. Kernel dependency estimation. Advances in Neural Information Processing Systems. The MIT Press, 2002.

[100] A. D. Wilson and A. F. Bobick. Parametric hidden markov models for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21 (9):884–900, 1999.

[101] S. Wold, H. Ruhe, H. Wold, and W. J. Dunne III. The collinarity problem in linear regression. the partial least squares (pls) approach to the generalized inverse. *SIAM Journal of Scientific and Statistical Computations*, 5:735–743, 1984.

[102] Y. Yacoob and M. J. Black. Parameterized modeling and recognition of activities. *Journal of Computer Vision and Image Understanding*, 73(2):398–413, 1999.

[103] Leon Zlajpah. Dexterity measures for optimal path control of redundant manipulators. In *Proceedings of International Conference on Automatisation and Control (ICAR)*, 1996.