# DIPLOMARBEIT

# Support Vector Machines for Regression Estimation and their Application to Chaotic Time Series Prediction

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs unter der Leitung von

**Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Franz Hlawatsch**

und

**Wiss.Mitarb.i.A. Dipl.-Ing. Michael Jachan**

als mitbetreuendem Assistenten
am Institut für Nachrichtentechnik und Hochfrequenztechnik der Technischen Universität Wien,
in Zusammenarbeit mit

**Directeur de recherche CNRS Patrick Flandrin**

und

**Chargé de recherche CNRS Patrice Abry**

am Institut für Physik der Ecole Normale Supérieure de Lyon,
eingereicht an der

**Technischen Universität Wien**
**Fakultät für Elektrotechnik und Informationstechnik**

von

**Herwig Wendt**
Finkenweg 7
4400 St. Ulrich

Wien, am 25.3.2005

# Kurzfassung

Support Vektor Maschinen (SVMs) sind eine relativ neue Funktionsschätzmethode und basieren auf dem Ansatz des überwachten Lernens. Sie verbinden verschiedene Techniken der statistischen Lerntheorie, der Optimierungstheorie, des maschinellen Lernens und der Kernfunktionen. In der vorliegenden Arbeit wird die Theorie der SVMs mit Schwerpunkt auf ihre Anwendung auf Regression behandelt. Weiters werden SVMs für Regression für die Vorhersage chaotischer Zeitreihen verwendet. Die Arbeit gliedert sich in drei Kapitel.

Im ersten Abschnitt werden die Teile der statistischen Lerntheorie, der Optimierungstheorie und des Konzepts der Kernfunktionen eingeführt, die gemeinsam die Theorie der SVMs bilden. Der Abschnitt ist modular aufgebaut und in einer Weise dargestellt, die einen Zugang auch für Lesern mit wenig Vorwissen im Bereich des maschinellen Lernens erlaubt. Dieser Abschnitt ergänzt die vorhandene Literatur über SVMs insofern, als einführende Literatur schwer zu finden ist und die theoretischen Aspekte nicht detailliert behandelt. Das Funktionsschätzproblem wird als Lernproblem eingeführt, für welches sich mit Konzepten der statistischen Lerntheorie lineare Lernmaschinen konstruieren lassen. Ergebnisse der Theorie von Lagrange führen diese Lernmaschinen in die eigentlichen Support Vektor Algorithmen über. Die Verwendung von Kernfunktionen ermöglicht es, auf reiche Klassen nichtlinearer Modellfunktionen zurückzugreifen. Wir behandeln im Detail die Herleitung der klassischen Support Vektor Algorithmen für Regression, sowie des $\nu$-Support Vektor Algorithmus als Beispiel für eine Weiterentwicklung der klassischen Algorithmen, der das automatische Ermitteln eines der in SVMs auftretenden Parameter ermöglicht. Im weiteren wird der Zusammenhang zwischen Maximum Likelihood Schätzern und der Wahl der Verlustfunktion aufgezeigt, und es wird dargestellt, wie SVMs mit anderen Funktionsschätzmethoden verbunden sind. Es wird gezeigt, wie Werte für die in den Algorithmen auftretenden Hyper-Parameter ermittelt werden können.

Der zweite Teil der Arbeit behandelt die Anwendung von SVMs für die Vorhersage von Zeitreihen mit chaotischem Verhalten. Die Eigenschaften von Chaos als einem Merkmal nichtlinearer deterministischer dynamischer Systeme werden diskutiert, und es wird dargestellt, wie skalare chaotische Zeitreihen durch das Rekonstruieren eines geeigneten Phasenraumes analysiert werden können. Praktische Methoden für die Phasenraumrekonstruktion werden angeführt. Es wird gezeigt, wie SVMs für die Vorhersage von chaotischen Zeitreihen in einem rekonstruierten Phasenraum verwendet werden können.

Im letzten Abschnitt werden die Vorhersage von chaotischen Zeitreihen mit SVMs, die durchgeführten numerischen Experimente sowie die verwendeten Zeitreihen beschrieben. Wir tragen neue numerischen Resultate für die Vorhersage der Hénon Zeitreihe, der Mackey-Glass Zeitreihe, der Lorenz Zeitreihe und der Santa Fe Data Set A Zeitreihe mit SVMs als globale und lokale Modelle mit unterschiedlichen Kernfunktionen bei. Die Arbeit endet mit einer Diskussion der Ergebnisse sowie einem ausführlichen Vergleich mit von anderen Autoren berichteten Ergebnissen zur Vorhersage chaotischer Zeitreihen.

# Abstract

Support vector machines (SVMs) are a quite recent supervised learning approach towards function estimation. They combine several results from statistical learning theory, optimisation theory, and machine learning, and employ kernels as one of their most important ingredients. The present work covers the theory of SVMs with emphasis on SVMs for regression estimation, and the problem of chaotic time series prediction. It is organised in three parts.

In the first part, the building blocks that contribute to the theory of SVMs are introduced. The necessary results from statistical learning theory, optimisation theory and kernels are summarised in a modular and self-contained way that makes them accessible as well to readers without background in these topics. The exposition complements already existing material on SVMs in so far as introductory literature that covers all theory employed by SVMs is hard to find. By viewing the function estimation problem as a learning problem, results from statistical learning theory allow the construction of linear learning machines. An application of Lagrangian theory casts these learning machines in forms that constitute the support vector algorithms for classification and regression estimation, which can employ rich classes of nonlinear modelling functions via the use of kernels. We consider in detail the derivation of the standard support vector algorithms for regression estimation and, as an example for a recently reported extension to the standard algorithms, describe the $\nu$-SVM which is capable of tuning one of the parameters involved in support vector training as a part of the training procedure. Then, the connection between maximum likelihood estimation and the choice of the loss function is established, and the introduction of kernels allows us to show how SVMs are related to other function estimation approaches. We finish the first part with a discussion on the important question of how the hyperparameters involved in the support vector algorithms can be assessed.

The second part considers the application of SVMs to chaotic time series prediction. The properties of chaos as a feature of nonlinear deterministic dynamical system are reviewed, and it is discussed how observed chaotic data can be analysed via phase space reconstruction and time delay embedding. Practical methods for time delay embedding are reviewed, and we show how SVMs can be applied to the function approximation problem arising from the task of predicting chaotic time series from embedded data.

In the third part, the prediction procedure with SVMs is described in detail. The setup of the numerical experiments and an analysis of the time series considered therein are given. We report and illustrate new results for chaotic time series prediction on the Hénon time series, the Mackey-Glass time series, the Lorenz time series and the Santa Fe data set A, obtained with SVMs as global and local models employing different kernel functions. We provide an exhaustive comparison with results reported by other authors. The work finishes with a discussion of the numerical experiments.

# Acknowledgements

First of all, I would like to thank Franz Hlawatsch for supervising this work. Many thanks as well to Michael Jachan for reading and discussing my manuscripts and for important input before fixing the final form of this work.

I would like to express my gratitude to Patrick Flandrin and Patrice Abry for many fruitful discussions and their sacrificing support during my work at ENS Lyon. Moreover, they together with Mr. Hlawatsch made this work possible in the first place. Thanks as well to my colleagues for contributing to this work in one way or another, and to the ENS Lyon for providing me with a work place.

I would like to thank my parents Heinrich und Hildegard, and my brother Gerhard, who gave me support during all my studies, and my girlfriend Faustine for her patience and support during my stay in France.

Last but not least, my stay at the ENS Lyon has been supported by the grant "Stipendium für kurzfristige wissenschaftliche Arbeiten im Ausland" of the Vienna University of Technology.

# Contents

# List of Figures

# List of Tables

# List of Symbols

| Symbol | |
|---|---|
| $a$, $A$ | scalar variable |
| $\mathbf{a}$ | vector |
| $\mathbf{A}$ | matrix |
| $[a_1, \cdots, a_n], [\mathbf{a}_1, \cdots, \mathbf{a}_n]$ | vector, matrix |
| $a_i$ | i-th coordinate of vector $\mathbf{a}$ |
| $A_{ij}$ | ij-th entry of matrix $\mathbf{A}$ |
| $\mathbf{a}^T, \mathbf{A}^T$ | transpose |
| $\langle \mathbf{a}_1, \mathbf{a}_2 \rangle$ | inner product |
| $\|\mathbf{a}\|$ | norm (Euclidean distance $\|\mathbf{a}\| = \sqrt{\langle \mathbf{a}, \mathbf{a} \rangle}$) |
| $\|\mathbf{a}\|_1$ | 1-norm ($\|\mathbf{a}\|_1 = \sqrt{\sum_j \|a_j\|}$) |
| $\|\mathbf{a}\|_2$ | 2-norm ($\|\mathbf{a}\|_2 = \sqrt{\sum_j a_j^2}$) |
| $\lambda_j$ | eigenvalues |
| $\boldsymbol{v}_j$ | eigenvectors |
| $A = \{\cdots\}$ | set of points or vectors |
| $\mathcal{A} = \{\cdots\}$ | function class, set of functions, function space |
| $p(\cdot)$ | probability density function |
| $P(\cdot)$ | cumulative density function, probability of an event |
| $p_{\text{emp}}(\cdot)$ | empirical density function |
| $E\{\cdot\}$ | expectation |
| $\hat{\sigma}_N^2$ | sample variance |
| $\bar{s}_N$ | sample mean |
| $I(\cdot)$ | mutual average information function |
| $C(\cdot)$ | linear autocorrelation function |
| $\delta(\cdot)$ | delta function |
| $\delta_{ij}$ | Kronecker symbol |
| $t$ | continuous time variable |
| $i, j, k, m, n$ | index variables |
| $t_s$ | sampling interval |
| $R(f)$ | expected risk of function $f$ |
| $R_{\text{emp}}(f)$ | empirical risk of function $f$ |
| $L(y, f(\mathbf{x})), L(y - f(\mathbf{x}))$ | loss function |
| $\mathcal{H}$ | base of functions contained in the hypothesis class |
| $y = h(\mathbf{x})$ | function from hypothesis space, modelling function |
| $S = \{(\mathbf{x}_n, y_n)\}$ | set of training examples |

| | |
|---|---|
| $N$ | number of training examples |
| $\mathbf{x}_n$ | n-th training vector |
| $y_n$ | n-th target value |
| $X$ | space of input vectors |
| $Y$ | space of target values |
| $n_x$ | dimension of input space |
| $\mathbf{w}, \boldsymbol{\theta}$ | weight vector |
| $b$ | constant offset |
| $C$ | trade-off parameter |
| $\nu$ | trade off parameter for $\nu$-SVR |
| $\epsilon$ | parameter of the $\epsilon$-insensitive loss function |
| $\xi_i, \check{\xi}_i$ | slack variables |
| $\boldsymbol{\xi}$ | vector of all slack variables |
| $\alpha_i, \check{\alpha}_i$ | Lagrange multiplier |
| $\boldsymbol{\alpha}$ | vector of all Lagrange multipliers |
| $\beta_i, \boldsymbol{\beta}$ | additional expansion coefficients |
| $\alpha^*$, $h^*$ | solution |
| $\mathcal{I}_{\text{sv}}$ | set of indices of support vectors |
| $n_{\text{NN}}$ | set of indices of nearest neighbours |
| $\boldsymbol{\Theta}$ | vector of hyperparameters |
| $\boldsymbol{\Gamma}$ | space of hyperparameters |
| $K(\cdot, \cdot)$ | Kernel function |
| $\mathbf{K}$ | Kernel matrix |
| $\boldsymbol{\phi}$ | map into feature space |
| $\phi_j$ | basis functions for feature map |
| $\mathcal{F}$ | feature space |
| $v$ | dimension of feature space |
| $\boldsymbol{f}$ | dynamical system |
| $\mathbf{x}(n)$ | state of dynamical system |
| $s(n)$ | scalar measurement of dynamical system |
| $\hat{s}(n)$ | estimate, prediction |
| $\Omega(\mathbf{x}(n))$ | measuring function |
| $\tau$ | embedding delay |
| $\mathbf{y}(n)$ | embedding vectors |
| $m$ | embedding dimension |
| $\mathcal{R}$ | time delay embedding space |
| $\mathcal{P}$ | state space |
| $D$ | dimension of state space |
| $d_A$ | attractor dimension |
| $P_{\tau_p}(\mathbf{y}(n))$ | $\tau_p$-step prediction function |
| $\tau_p$ | prediction step size |
| $M$ | number of available embedding vectors |
| $N_E$ | number of training examples for time series prediction |

# Chapter 1

# Introduction

In general, estimation and approximation applications involve making inference from observations that are distorted or corrupted in some unknown manner, when the information that one wishes to extract is unknown to the observer. The simplest way to approximate a function would be to take the mean of the observations. Choosing linear functions or more complicated bases of functions would be a more sophisticated approach, and the solution to obtain better results seems to be to increase the complexity of the base. This is not true since one encounters the well-known effect of overfitting, which means that the complexity of the system of functions used is too high. For obtaining good approximations, one needs to take the complexity of the base of functions into account.

There already exists a large set of approximation approaches, for instance splines and methods based on decomposition into orthogonal systems. All these methods suffer from shortcomings that are tried to be overcome by the support vector approach. Splines and decomposition approaches share the problem of exponential increase in the number of coefficients with the dimensionality of the problem. One solution is to use nonseparable expansions, e.g. neural networks, which allow tractable solutions of high-dimensional problems. Their architecture has to be defined a priori or modified by some heuristics during training, which cannot assure that the optimal structure of the network is found for a particular problem. Moreover, the possibilities for controlling the complexity of the function base are rather limited, and the training algorithm can get stuck in local minima. Only for the asymptotic case and for the case of known prior probabilities optimal selection criteria have been obtained.

In contrast, support vector machines (SVMs) possess a number of advantages. Their architecture does not have to be determined beforehand, and input data of any arbitrary dimension can be treated with only a linear cost in the number of input dimensions. Moreover, the training has a unique solution, and the modelling functions may be chosen within a rich function base having to satisfy only some conditions from functional analysis. Capacity is controlled efficiently by implementing a learning bias that involves a regularisation term [59]. SVMs are a rather new concept in learning theory. Although its origins can be dated back the 60's, it attracted attention only after Vapnik's and Cortes' work in 1995 [12] [67]. Since then, SVMs proved excellent performance in many applications such as pattern recognition, text categorisation and solution of inverse problems. Many aspects in the theory of SVMs are still under intensive research, and the number of introductory literature is limited. We decide to give a thorough introduction to SVMs with emphasis on regression estimation, ranging from basic terminology and mathematical tools to more

advanced topics and to questions arising in practise, and apply SVMs to the important problem of chaotic time series prediction.

## Scope and Organisation

The goal of this work twofold. First, we aim to give a comprehensive overview of the theory of SVMs with emphasis on regression estimation that is accessible for readers which are not familiar with terminology from machine learning, statistical learning theory, optimisation theory and kernels. Second, we want to demonstrate how SVMs can be applied to the specific problem of chaotic time series prediction and compare their performance to other approximation approaches. Although we attempt to hold the exposition as self-contained as possible, a detailed treatment of all aspects in connection with SVMs would go beyond the scope of this work. In particular, we will not cover implementational issues, and we will omit all proofs. The interested reader is referred to the literature.

The work is organised in three chapters which can be read in a modular fashion. In the first chapter, the theory of SVMs is summarised. In the second chapter, we introduce the problem of chaotic time series prediction and show how SVMs can be applied to this problem. In the third chapter, numerical results for prediction on four chaotic time series are reported.

Chapter 2 is divided into six modules. In Section 2.1, we introduce the reader to statistical learning theory. We give ourselves satisfied with the classical concepts of risk minimisation and Vapnik Chervonenkis theory and omit recent developments such as data dependent structural risk minimisation and the luckiness concept, as these are still subject to active research. If we need results that emerge from beyond the classical concepts, we quote them and refer the reader to literature. In Section 2.2, we introduce the concept of margin and shortly talk about SVMs for classification, and then bring all concepts together and derive the support vector algorithms for regression estimation in Section 2.3. At this point, we already need techniques from optimisation theory and kernels. Since the key idea of SVMs is contributed by statistical learning theory, we move a self-contained exposition on optimisation theory and kernels to Sections 2.4 and 2.5. Readers not familiar with these topics may take a look onto these sections already before reading Section 2.2 and 2.3. In Section 2.6, we address the important question of how the parameters involved in the SVM algorithms can be assessed in practise. A discussion on how SVMs are related to other function estimation techniques can be found at the end of Section 2.3, where we establish the connection between maximum likelihood estimation and loss functions, and at the end of Section 2.5, where we show how SVMs are related to Gaussian processes.

In Chapter 3, a short introduction to chaos in dynamical systems is given. The problem of embedding observed chaotic data in practice is discussed, and it is shown how SVM can be applied to chaotic time series prediction.

Chapter 4 describes in detail the SVM models and the time series we consider for prediction, and summarises the experimental setup we use for obtaining our results. The prediction results for the Hénon time series, the Mackey-Glass time series, the Lorenz time series and the Santa Fe Data Set A are discussed and illustrated and compared with results reported in literature.

# Chapter 2

# Theory of Support Vector Machines

## 2.1 Statistical Learning Theory

The basic problem we are dealing with in this work is the problem of approximating or estimating a function from given data. In difference to approximation, where we know that our data are correct and we want to obtain a function that (up to a certain precision) produces the same values as our measurements, in the case of function estimation the data may be corrupted by noise, and in many cases we may not even know the noise model. Both problems can be treated with the same formalism by choosing an appropriate loss function (see Section 2.3.10).
Out of the considerable body of theory that has been developed in statistical learning theory, we briefly review some concepts and results necessary for the development of support vector learning algorithms. For details, we refer to [67] and [68], on which this section is mainly based.

### 2.1.1 Problem Setting

We consider the problem of estimating a desired dependency using only a limited number of observations.

#### Learning Problem

Suppose we are given $N$ i.i.d. samples, the *training set*

$$S = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}, \tag{2.1}$$

where each sample (observation) consists of a vector $\mathbf{x}_n \in X \subseteq \mathbb{R}^{n_x}$ of attributes from the input space $X$ and a label or target[1] $y_n \in Y$, and a class of functions $\mathcal{H}$, called the *hypothesis space*

$$\mathcal{H} = \{h | X \subseteq \mathbb{R}^{n_x} \mapsto Y\}.$$

The functions $h \in \mathcal{H}$, $y = h(\mathbf{x})$ are called *hypothesis functions* or *hypotheses*, and $\mathcal{H}$ defines a *learning machine*. The training set (2.1) is drawn according to some fixed but *unknown probability distribution* $P(\mathbf{x}, y)$.

[1] Throughout this work, we will consider the space of target values to be either $Y = \{-1, 1\}$ or $Y \subseteq \mathbb{R}$.

The learning problem is now to choose the function $h^* \in \mathcal{H}$ that has lowest error on unseen samples drawn from $P(\mathbf{x}, y)$. More precisely, by penalizing deviations by means of a *loss function* $L(y, h(\mathbf{x}))$, with $L(\cdot, \cdot) \geq 0$ and $L(z, z) = 0$, the solution to the learning problem is the hypothesis $h^* \in \mathcal{H}$ that minimises the expected value of the *loss* or *risk*,

$$R(h) = \mathrm{E}_{P(\mathbf{x},y)}\{L(y, h(\mathbf{x}))\} = \int L(y, h(\mathbf{x}))dP(\mathbf{x}, y), \tag{2.2}$$

$$h^* = \arg\min_{h \in \mathcal{H}} R(h).$$

Thus, the goal of learning is to select $h \in \mathcal{H}$ that minimises (2.2) in the situation where the joint probability function $P(\mathbf{x}, y)$ is unknown and the only information is contained in the training set (2.1) drawn i.i.d. from $P(\mathbf{x}, y)$. Notice that within the setting of the learning problem, it is not necessary that $\mathcal{H}$ contains the target function generating the training set (2.1), possibly corrupted by noise.

**Example:** Binary classification.
Binary classification is performed by using some function $h(\mathbf{x})$ such that the input $\mathbf{x}$ is assigned to the positive class if $h(\mathbf{x}) \geq 0$, and to the negative class otherwise. Let $\mathcal{H} = \{h|\mathbb{R}^{n_x} \mapsto \{-1, 1\}\}$ and $Y = \{-1, 1\}$ such that a sample $(\mathbf{x}_n, y_n)$ belongs to the positive class if $y_n = 1$, and to the negative class otherwise. For the loss function

$$L(y_n, h(\mathbf{x}_n)) = \begin{cases} 0 & y_n h(\mathbf{x}_n) > 0 \\ 1 & y_n h(\mathbf{x}_n) \leq 0, \end{cases}$$

(2.2) gives the probability of classification error. The learning problem is therefore to find the hypothesis that minimises the classification error when $P(\mathbf{x}, y)$ is unknown but the data (2.1) is given.
As a simple classification example, consider the training set $S_1 = \{(x_n, y_n)\} = \{(-1.5, 1), (-1.1, -1), (0.3, -1), (0.9, -1), (1.4, 1), (1.8, 1)\}$ with $(x_n, y_n) \in \mathbb{R} \times \{-1, 1\}$. A possible hypothesis would be

$$h(\mathbf{x}_n) = \begin{cases} +1 & |\mathbf{x}_n| \geq 1.2 \\ -1 & \text{otherwise.} \end{cases}$$

This hypothesis produces no classification error on the training set $S_1$, but we cannot say whether it is the minimiser of (2.2).

The property of a function to have low error on unseen samples is often referred to as *generalisation*, and the expectation of the error of a hypothesis on unseen examples drawn i.i.d. from $P(\mathbf{x}, y)$, the risk, as *generalisation error*.

### 2.1.2 Risk Minimisation

Equation (2.2) is the fundamental problem to solve in statistical learning theory. Unfortunately, the risk cannot be minimised directly, as the underlying probability distribution $P(\mathbf{x}, y)$ is unknown. What we can do is to try to choose a hypothesis that is *close* to the optimal one, based on the available information (the training set (2.1) and the properties of $\mathcal{H}$). For that, we need an induction principle.

**Empirical Risk Minimisation (ERM) Induction Principle**

Approximating the unknown probability distribution by the empirical probability density function [60]

$$p_{\text{emp}}(\mathbf{x}, y) = \frac{1}{N}\sum_{n=1}^{N}\delta(\mathbf{x}-\mathbf{x}_n)\delta(y-y_n),$$

leads to the *empirical risk functional*

$$R_{\text{emp}}(h) \triangleq \int_{X\times Y} L(y, h(\mathbf{x}))p_{\text{emp}}(\mathbf{x}, y)dxdy = \frac{1}{N}\sum_{n=1}^{N} L(y_n, h(\mathbf{x}_n)), \tag{2.3}$$

measuring the mean error rate on the training set (2.1). In using the *Empirical Risk Minimisation Induction Principle* (ERM Principle), we approximate the risk (2.2) by the empirical risk (2.3):

**Definition 1 (ERM Principle).** *Replace the risk functional (2.2) by the empirical risk functional (2.3). Then, approximate the hypothesis $h^*$ minimising (2.2) by the hypothesis $h^*_{\text{ERM}}$ minimising (2.3),*

$$h^*_{\text{ERM}} = \arg\min_{h\in\mathcal{H}} R_{\text{emp}}(h).$$

It is possible to give conditions on $\mathcal{H}$ under which the empirical risk (2.3) converges asymptotically ($N \to \infty$) to the risk (2.2). However, for a small sample size $N$, large deviations may occur[2]. What is more, trying to minimise the empirical risk (2.3) generally

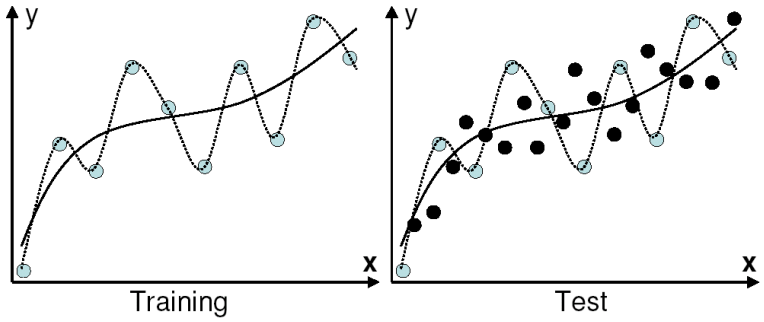


Figure 2.1: *The overfitting dilemma: Given only a limited number of data, both the dotted and the solid hypothesis might be true. The dotted hypothesis has lower error on the training set, and the solid hypothesis has lower capacity. The empirical risk minimisation induction principle would therefore select the dotted hypothesis, and the solid one would be chosen by the structural risk minimisation induction principle introduced below (left). The dotted hypothesis is subject to overfitting, i.e. it is too complex (right).*

leads to an ill-posed problem [66], i.e., the map from $S$ to $h$ may be discontinuous. This results in a phenomenon well-known as *overfitting* (Figure 2.1), i.e., the selected hypothesis may be too complex. One way to avoid overfitting is to restrict the capacity of the function class $\mathcal{H}$, typically by introducing a *regularisation* term [66]. This raises the problem of how to find the optimal capacity of the function class, known as model selection.

[2]Consider e.g. choosing $h$ such that $h(\mathbf{x}_n) = y_n$ for $(\mathbf{x}_n, y_n) \in S$ and $h(\mathbf{x}) = 0$ for $(\mathbf{x}, y) \in X\backslash S$. This function has zero empirical risk, but it is unlikely that it is the minimiser of (2.2).

**Structural Risk Minimisation (SRM) Induction Principle**

Within the framework of statistical learning theory, a remarkable family of bounds on the risk (2.2) has been derived, governing the relation between the capacity of a function class and its performance in terms of generalisation. The theory grew out of consideration under which circumstances and how quickly the mean of some empirical quantity converges to the true mean as the number of data points increases. These bounds originally motivated SVMs, as introduced by Cortes and Vapnik [12]. The bounds typically consist of an empirical risk term $R_{\text{emp}}(h)$ and a confidence term $\Phi$ that grows monotonically with increasing capacity $\zeta$ of the function base used. Roughly speaking, the capacity measures how flexible or complex a function base is. For instance, linear functions are of lower capacity than polynomials. One particular capacity measure $\zeta$ of a function class is the *Vapnik-Chervonenkis dimension* (VC dimension)[3] $d$ (e.g. [69]). The VC dimension measures how many vectors we can correctly classify with a function in all possible ways of labelling.

> **Example:** VC dimension of hyperplanes.
> Given $n_x + 1$ vectors $\{\mathbf{x}_1, \cdots, \mathbf{x}_{n_x+1}\}$ in $n_x$-dimensional space $\mathbb{R}^{n_x}$, we can assign labels $\{y_1, \cdots, y_{n_x+1}\}$, $y \in \{-1, 1\}$ to them in $2^{n_x+1}$ different combinations. There exists a set of $n_x + 1$ such vectors for which we can find a hyperplane $h(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b$, $\mathbf{x}, \mathbf{w} \in \mathbb{R}^{n_x}$ for each $2^{n_x+1}$ possible way of labelling that *separates* the vectors with positive labels from the vectors with negative labels, i.e. we can find hyperplanes such that the vectors with positive lables are on the one side of them, and the vectors with negative labels are on the other side (Figure 2.2). We say that the hyperplanes in $\mathbb{R}^{n_x}$ *shatter* the $n_x + 1$ vectors $\{\mathbf{x}_1, \cdots, \mathbf{x}_{n_x+1}\}$. Therefore, the VC dimension of hyperplanes in $\mathbb{R}^{n_x}$ is equal to $d = n_x + 1$ [67].

**Definition 2 (VC dimension).** *The VC dimension of a set of indicator functions $\mathcal{H}$ is $d$ if there exists a set of $d$ vectors $\mathbf{x}_1, \cdots, \mathbf{x}_d$ which can be separated in all $2^d$ possible ways of labelling using functions of this set (is shattered by $\mathcal{H}$), and there exists no larger set of vectors that can be shattered by $\mathcal{H}$. If for any $d$ there exists a set of $d$ vectors which can be shattered by $\mathcal{H}$, then the VC dimension is equal to infinity.*

*The VC dimension $d$ of the set $\mathcal{H}$ of real-valued functions $h$ is defined to be the VC dimension of the set of indicator functions*

$$f(h(\mathbf{x}), \beta) = \begin{cases} 0 & h(\mathbf{x}) - \beta < 0 \\ 1 & h(\mathbf{x}) - \beta \geq 0, \end{cases}$$

$$d\left(h(\mathbf{x})\right) \stackrel{\triangle}{=} d\left(f(h(\mathbf{x}), \beta)\right).$$

The bounds provided by statistical learning theory are of the form

$$R(h) \leq R_{\text{emp}}(h) + \Phi(N, \zeta, \delta), \tag{2.4}$$

[3] The bounds formulated in terms of the VC dimensions are only the last elements in a series of tighter bounds formulated in terms of other capacity measure concepts.

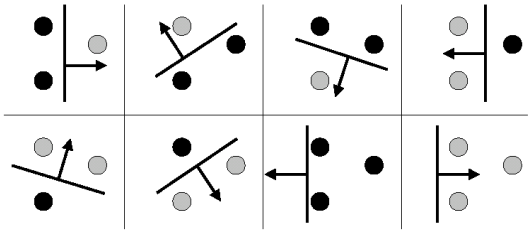

Figure 2.2: *Illustration of VC dimension of hyperplanes. In $n_x = 2$-dimensional input space $X \subseteq \mathbb{R}^2$, a maximum of $d = n_x + 1 = 3$ vectors can be shattered by hyperplanes, i.e. they can be separated by hyperplanes in all $2^d = 8$ possible ways of labelling.*

where $N$ is the size of the training set (2.1), $1 - \delta$ is the probability by which the bound holds true, and $\zeta$ is the capacity of the function class $h$ belongs to (e.g. its VC dimension $d$). The confidence term $\Phi(N, \zeta, \delta)$ grows monotonically with increasing capacity $\zeta$. A constructive example of such a bound will be given in Section 2.2. Remarkably, these bounds on the risk are all independent of the probability distribution $P(\mathbf{x}, y)$ and can therefore be used for our learning problem. What is more, they are in themselves independent of the dimensionality of the input space $X$ and can thus address the *curse of dimensionality* (see e.g. [41]).

Thus, with such a bound on the risk at hand, we can replace the ERM induction principle and approximate the minima of the risk (2.2) with the minima of a bound (2.4) on the risk. This is formulated in the *structural risk minimisation induction principle (SRM principle).*

**Definition 3 (SRM Principle).** *Introduce a structure in $\mathcal{H}$ by dividing $\mathcal{H}$ into a nested sequence of hypothesis classes $\mathcal{H}_1 \subset \mathcal{H}_2 \subset \cdots \subset \mathcal{H}_m \subset \cdots \subset \mathcal{H}_M$, such that the capacity measure $\zeta_m$, e.g. the VC dimension $d_m$, of each subset can be computed or bounded. Then the capacities of the subclasses satisfy $\zeta_1 \leq \zeta_2 \leq \cdots \leq \zeta_m \leq \cdots \leq \zeta_M$. Choose the hypothesis subclass $\mathcal{H}_m$ for which the bound (2.4) on the risk functional is minimal. The function $h^*$ minimising the risk (2.2) is approximated by*

$$h^*_{\text{SRM}} = \arg \min_{h \in \mathcal{H}_m} R_{\text{emp}}(h).$$

Therefore, the SRM principle suggests a trade off between the quality of approximation and the complexity of the approximating function. For any distribution function, the SRM method provides convergence to the best possible solution with probability one [69]. An illustration of the SRM principle is given in Figure 2.3.

### 2.1.3 Constructing a Learning Machine

One can think of two strategies for minimising bounds of type (2.4):

1. *Keep the confidence term $\Phi(N, \zeta, \delta)$ fixed by choosing an appropriate construction of a machine and minimise the empirical risk $R_{\text{emp}}(h)$.*
   This is equivalent to ERM and is implemented by neural networks, where one defines a network structure beforehand and then minimises the empirical risk.

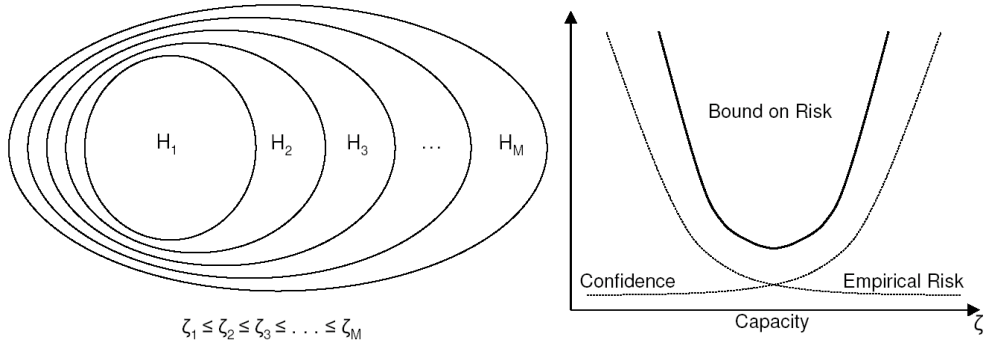


Figure 2.3: *Illustration of the structural risk minimisation inductive principle. A nested sequence of hypothesis, with capacity measure $\zeta$ (left). The bound on the risk, determined on the one hand by the empirical risk, and on the other hand by the capacity $\zeta$ of the hypothesis class that decides on the value of the confidence term $\Phi(N, \zeta, \delta)$ (right). The SRM solution is found as the hypothesis having lowest empirical risk within the hypothesis subclass producing the lowest value of the bound on the risk.*

2. *Minimise $CR_{\text{emp}}(h) + \Phi(N, \zeta, \delta)$, a trade off between empirical risk and confidence term, measured by a regularisation parameter $C$.*
   This is equivalent to SRM[4] when used over a nested sequence of hypothesis classes.

The SRM principle provides us with a powerful induction principle for constructing learning machines for the general learning problem of Section 2.1.1. Each particular structure on a particular hypothesis space gives rise to a learning algorithm, consisting of performing SRM in the given structure of sets of functions. Support vector learning machines employ the particular hypothesis space of linear functions in some high-dimensional feature space implicitly defined by a *kernel* (see Section 2.5). Soft margin SVMs follow strategy 2 directly, whereas Hard margin SVMs use strategy 2 with $C \to \infty$ and thus, do not allow for training errors.

## 2.2 Support Vector Classification (SVC)

Historically, it has been the classifiers that paved the way for SVMs. Their learning bias has been derived from a bound on the generalisation by Vapnik [67], by exploiting a geometric concept called the *margin*. These results from statistical learning theory have already been known at the time of the introduction of support vector machines for classification (SVC). In contrast, support vector machines for regression estimation (SVR) have been introduced by translating the learning bias employed by SVC for loss functions suitable for regression estimation. Since Vapnik's bound does not apply to loss functions specific for the regression task, SVR have not possessed a clean motivation or interpretation at the time of their introduction. It was only later that generalisation bounds applicable for SVR have been found, justifying their induction principle. Whereas SVC allow for an intuitive and illustrative interpretation of their learning bias in terms

[4]One can view $C$ as an additional linear scaling in the loss function, such that $L'(y, h(\mathbf{x})) = CL(y, h(\mathbf{x}))$.

of the *maximal margin hyperplane*, this is not the case for SVR, where the margin plays only a technical role.

We decide to follow the original introduction of the SVM and summarise SVC in this section. This is done for the sake of completeness of the exposition on SVMs, and for the following reasons.

- The interpretation of the SVC algorithms and solution provides an intuitive insight into the nature of this type of learning machine.
- SVC and SVR have properties in common that can be seen as characteristic features of all types of SVMs.
- The learning bias of SVR can be motivated through an argument involving SVC.
- Most of the literature on SVMs is mainly concerned with SVC.

Apart from the present section, the rest of this work will only be concerned with SVR. A self-contained introduction of support vector classification would therefore go beyond the scope of this work. We refer to e.g. [5], [13] and [23] for a more detailed treatment of this subject, and will mainly follow these works.

In the present and in the following section, we will need results from optimisation theory that can be found in Section 2.4.

### 2.2.1 The Classification Problem

Suppose we are given the training data

$$S = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}, \quad \mathbf{x}_n \in X \subseteq \mathbb{R}^{n_x},\, y_n \in \{-1, +1\}, \tag{2.5}$$

with samples $(\mathbf{x}_n, y_n)$ that are drawn i.i.d. according to some unknown but fixed probability distribution $P(\mathbf{x}, y)$. Let the hypothesis space be fixed to the class of linear indicator functions in the $n_x$-dimensional space (*hyperplanes*)[5],

$$\mathcal{H} = \{h | h(\mathbf{x}) = \text{sign}(\langle \mathbf{w}, \mathbf{x} \rangle + b); \quad \mathbf{w}, \mathbf{x} \in \mathbb{R}^{n_x}; b \in \mathbb{R}\}, \tag{2.6}$$

where

$$\text{sign}(\Delta) = \begin{cases} +1 & \Delta > 0 \\ -1 & \Delta \leq 0. \end{cases}$$

For the 0/1 loss function

$$L_{0/1}(h(y_n, \mathbf{x}_n)) = \begin{cases} 0 & y_n h(\mathbf{x}_n) > 0 \\ 1 & y_n h(\mathbf{x}_n) \leq 0, \end{cases} \tag{2.7}$$

the risk (2.2) gives the probability that $\hat{y} = h(\mathbf{x})$ differs from $y$ (classification error). As discussed in the previous section, the learning problem consists now of finding the

[5]The *inner product* $\langle \cdot, \cdot \rangle$ is defined as $\langle \mathbf{x}, \mathbf{z} \rangle = \mathbf{x}^T \mathbf{z} = \sum_i x_i z_i$.

hypothesis that minimises the classification error when $P(\mathbf{x}, y)$ is unknown but the training set $S$ is given.

The choice of hypothesis space we consider may seem very limiting. We will, however, see that the algorithms derived for this setting can easily and efficiently be generalised to a rich class of nonlinear functions via the use of kernels (Section 2.5).

### 2.2.2 A Bound on the Generalisation Performance of a Two Class Classifier

In order to use the SRM principle (Definition 3) for the classification problem, we need to have a constructive bound on the risk at hand. Vapnik proposed the following bound on the generalisation error that applies for our specific problem setting. The bound holds true with probability $1 - \delta$. It employs the VC dimension $d$ as capacity measure and has originally motivated the SVMs as introduced in [12]. Its derivation is not trivial and would go beyond the scope of this work [67].

**Theorem 4.** *Let $d$ denote the VC dimension of the function class $\mathcal{H}$ and let $R_{\text{emp}}$ be defined by (2.3), using a 0/1 loss function defined by (2.7). For all $\delta > 0$ and $h \in \mathcal{H}$, the inequality bounding the risk (2.2)*

$$R(h) \leq R_{\text{emp}}(h) + \sqrt{\frac{d\left(\ln \frac{2N}{d} + 1\right) - \ln \frac{\delta}{4}}{N}} \tag{2.8}$$

*holds with probability at least $1 - \delta$ for $N > d$.*

This bound on the risk is of form (2.4), containing the empirical error $R_{\text{emp}}$ and a confidence term $\Phi(N, d, \delta)$ as additive terms. The confidence term grows monotonically with increasing VC dimension $d$. Thus, if we are able to calculate or bound the VC dimension of the class $\mathcal{H}$ of indicator functions in $\mathbb{R}^{n_x}$, we can bound the expected error by (2.8). If we in addition find a concept that allows us to divide $\mathcal{H}$ into a nested sequence of hypothesis subclasses $\mathcal{H}_1 \subset \cdots \subset \mathcal{H}_m \subset \cdots \subset \mathcal{H}_M$ such that $d_1 \leq \cdots \leq d_m \leq \cdots \leq d_M$, we can perform SRM and select the hypothesis $h^* \in \mathcal{H}_m$ for which (2.8) is minimal,

$$\begin{aligned}
&R_{\text{emp}}(h^*) + \sqrt{\frac{d_m\left(\ln \frac{2N}{d_m} + 1\right) - \ln \frac{\delta}{4}}{N}} \leq R_{\text{emp}}(h_i) + \sqrt{\frac{d_i\left(\ln \frac{2N}{d_i} + 1\right) - \ln \frac{\delta}{4}}{N}}, \\
&R_{\text{emp}}(h^*) \leq R_{\text{emp}}(h_m), \\
&\qquad h^*, h_m \in \mathcal{H}_m, \quad h_i \in \{\mathcal{H}_1, \cdots, \mathcal{H}_{m-1}, \mathcal{H}_{m+1}, \cdots, \mathcal{H}_M\}.
\end{aligned}$$

Note that alternative concepts for measuring capacity, giving rise to tighter bounds on the risk functional than bounds employing the VC dimension $d$, can be found (e.g. [13]). We will make use of such bounds for motivating the soft margin support vector algorithms later on in this section and the support vector algorithms for regression estimation in Section 2.3.

### 2.2.3 Constructing the Learning Machines for Classification

The bound (2.8) on the risk of a linear classifier grows monotonically with increasing VC dimension $d$. SVMs employ hyperplanes (2.6) in some feature space, and we have seen in the previous section that the VC dimension of hyperplanes in $n_x$-dimensional space is $d = n_x + 1$ and thus prescribed by the dimensionality of the input space $X$. Therefore, we cannot perform SRM directly on this function class, since the SRM principle requires to introduce a structure on it, for which we need to be able to control its capacity.

The key to performing SRM is to restrict the flexibility of the hyperplanes (2.6) by forcing them to have a certain *margin* on the training set, a concept we will introduce shortly. It can be shown that the size of the margin bounds the value of the VC dimension of the restricted hyperplanes independently of the dimension of the space $X$ they live in. By controlling the size of the margin, we can control the VC dimension of the restricted hyperplanes, which allows us to introduce a structure on $\mathcal{H}$ and perform SRM with bound (2.8).

#### Hard Margin Classifier

Suppose there exists a hyperplane $h \in \mathcal{H}$ that separates the positive samples of the training set (2.5) from the negative ones without error on the training set ($R_{\text{emp}} = 0$). We call the distance of a hyperplane to the training vector that lies closest to it the *margin*.

**Definition 5 (Margin).** *The functional margin of a sample* $(\mathbf{x}_n, y_n)$ *with respect to a hyperplane* $(\mathbf{w}, b)$ *is the quantity* $\gamma_n = y_n \left(\langle \mathbf{w}, \mathbf{x}_n \rangle + b\right)$.
*The functional margin of a hyperplane* $(\mathbf{w}, b)$ *with respect to a training set (2.5) is the minimum of the margins of the samples of the training set,*

$$\gamma_S \stackrel{\triangle}{=} \min_{(\mathbf{x}_n, y_n) \in S} y_n \left(\langle \mathbf{w}, \mathbf{x}_n \rangle + b\right).$$

*The equivalent quantity for hyperplanes with unit weight vector is called geometric margin,*

$$\gamma \stackrel{\triangle}{=} \min_{(\mathbf{x}_n, y_n) \in S} y_n \left(\langle \mathbf{w}, \mathbf{x}_n \rangle + b\right), \quad \|\mathbf{w}\| = 1.$$

The geometric margin of a hyperplane with respect to a training set is illustrated in Figure 2.4 (left).

We note that in the definition of linear classifiers (2.6) lies an inherent degree of freedom, as we can rescale the weight vectors $\mathbf{w}$ without changing the functionality of the classifier. This can be used to express the geometric margin of a hyperplane in terms of the norm of its weight vector. By fixing the functional margin of the hyperplane to be $\gamma_S = 1$ for training vectors[6] $\mathbf{x}^+$ and $\mathbf{x}^-$ (*canonical hyperplane*),

$$y_n \left(\langle \mathbf{w}, \mathbf{x}_n \rangle + b\right) \geq 1, \quad n = 1, \cdots, N$$

the geometric margin $\gamma$ with respect to the training set $S$ is

$$\gamma = \frac{1}{2}\left(\langle \frac{\mathbf{w}}{\|\mathbf{w}\|}, \mathbf{x}^+ \rangle - \langle \frac{\mathbf{w}}{\|\mathbf{w}\|}, \mathbf{x}^- \rangle\right) = \frac{1}{2\|\mathbf{w}\|}\left(\langle \mathbf{w}, \mathbf{x}^+ \rangle - \langle \mathbf{w}, \mathbf{x}^- \rangle\right) = \frac{1}{\|\mathbf{w}\|}.$$

[6] For convenience of notation, we use the superscripts $+/-$ for samples lying closest to the hyperplane and belonging to the positive/negative class, respectively.

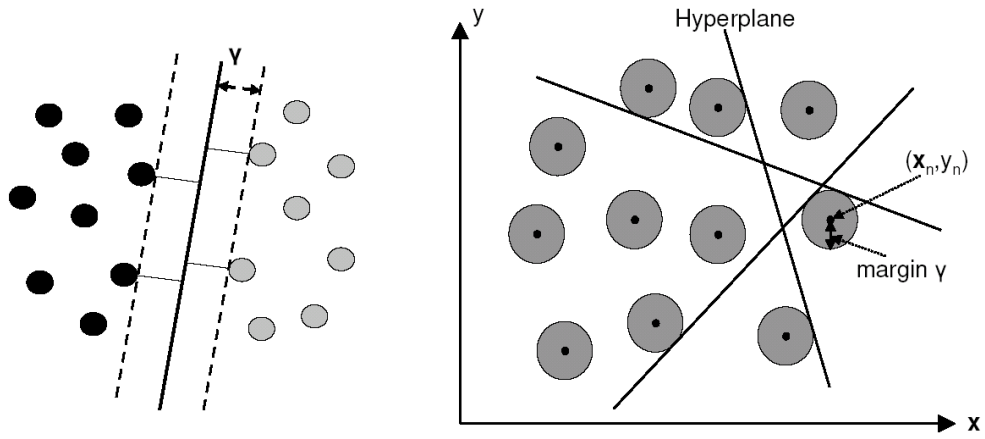


Figure 2.4: *The geometric margin $\gamma$ of a hyperplane with respect to a linearly separable data set measures the distance between the hyperplane and the sample or samples that lie closest to it (left). Introducing a margin clearly restricts the flexibility - and therefore the capacity - of hyperplanes (right).*

It can now be shown that the VC dimension of canonical hyperplanes that are restricted to have a geometric margin of at least $\gamma = \frac{1}{\|\mathbf{w}\|}$ on the training set is bounded by the following theorem [67].

**Theorem 6.** *A subset of canonical hyperplanes defined on $\mathbb{R}^{n_x}$ and satisfying the constraint $\|\mathbf{w}\| = \frac{1}{\gamma} \leq A$ has the VC dimension $d$ bounded by the inequality*

$$d \leq \min\left(R^2 A^2, n_x\right) + 1,$$

*where $R$ is the radius of a sphere containing all training vectors.*

The influence of the margin on the capacity of hyperplanes is illustrated in Figure 2.4 (right).

Theorem 6 allows us to control the capacity of the hyperplanes by constraining $\|\mathbf{w}\| \leq A_m$. Therefore, we can divide the hypothesis space (2.6) into a nested sequence of hypothesis subclasses $\mathcal{H}_1 \subset \cdots \subset \mathcal{H}_m \subset \cdots \subset \mathcal{H}_M$ by setting $A_1 \leq \cdots \leq A_m \leq \cdots \leq A_M$ and perform SRM. Since the hard margin classifier requires the empirical risk to be zero, performing SRM consists in this case of simply choosing the hyperplane that has smallest norm $\|\mathbf{w}\|$ and produces no training error. This hyperplane has the smallest VC dimension of all possible hyperplanes, and therefore minimises the bound on the risk (2.8) with $R_{\text{emp}} = 0$. We call such a hyperplane $(\mathbf{w}, b)$ that separates the training set without error ($R_{\text{emp}} = 0$) and that has maximal geometric margin $\gamma$ the *optimal separating hyperplane*. Its parameters $(\mathbf{w}^*, b^*)$ are found as the solution of the following quadratic programme.

**Proposition 7 (Optimal Separating Hyperplane).** *Given the training set $S = ((\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N))$, $\mathbf{x}_n \in \mathbb{R}^{n_x}$, $y_n \in \{-1, +1\}$, the hyperplane $(\mathbf{w}, b)$ that solves the quadratic programme*

$$\min_{\mathbf{w} \in \mathbb{R}^{n_x}, b \in \mathbb{R}} \langle \mathbf{w}, \mathbf{w} \rangle \quad \text{subject to } y_n \left( \langle \mathbf{w}, \mathbf{x}_n \rangle + b \right) \geq 1; \quad n = 1, \cdots, N$$

*is the hyperplane separating $S$ with maximal geometric margin $\gamma = \frac{1}{\|\mathbf{w}\|}$ [13].*

**Soft Margin Classifier**

If the training set is not linearly separable, Proposition 7 does not have a solution, as its feasible region is empty. What is more, the maximal margin measure is sensitive to noise, since single outliers can force the solution to have small margin. Then, the optimal separating hyperplane cannot be the minimiser of bound (2.8), and we need a more robust concept that allows for empirical risk $R_{\text{emp}} \neq 0$ [12].

In order to allow for errors on the training set, *slack variables* $\xi_n \geq 0$ are introduced,

$$\xi((\mathbf{x}_n, y_n), h, \gamma) = \xi_n \stackrel{\triangle}{=} \max(0, \gamma_S - y_i h(\mathbf{x}_n)),$$

measuring by how much a training sample $(\mathbf{x}_n, y_n)$ fails to have functional margin $\gamma_S$ (Figure 2.5). Note that $\xi_n > \gamma_S$ implies misclassification of $(\mathbf{x}_n, y_n)$. Then, for $\sigma \to 0$, the

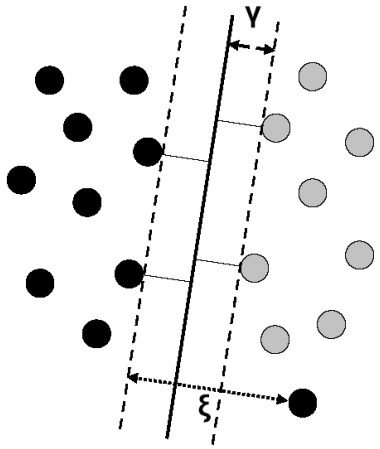


Figure 2.5: *The introduction of slack variables $\xi$ allows for errors on the training set. They measure by how much a training vector fails to have functional margin $\gamma_S$.*

empirical risk is approximated by[7]

$$F_\sigma(\boldsymbol{\xi}) \stackrel{\triangle}{=} \frac{1}{N} \sum_{n=1}^{N} \xi_n^\sigma \underset{\sigma \to 0}{=} R_{\text{emp}}.$$

Only the choices $\sigma = 1$ (1-norm) and $\sigma = 2$ (2-norm) are computationally efficient and lead to a quadratic programme. For this choices of $\sigma$, $F_\sigma(\boldsymbol{\xi})$ is only an upper bound on $R_{\text{emp}}$.

Analogous to the hard margin classifier, we can define a generalised separating hyperplane as the hyperplane that has maximal margin on the reduced training set $S'$ where all the samples having non-zero $\xi_n$ have been removed (*soft margin*). Due to Theorem 6, we can again introduce a structure $\mathcal{H}_1 \subset \cdots \subset \mathcal{H}_m \subset \cdots \subset \mathcal{H}_M$ on these hyperplanes by forcing $\|\mathbf{w}\| < A_m$, $A_1 \leq \cdots \leq A_m \leq \cdots \leq A_M$ and select the hyperplane $h^* \in \mathcal{H}_m$ that produces the lowest bound (2.8) on the risk by means of the SRM principle. This hyperplane $h^*$ is called the *generalised optimal separating hyperplane*. Notice that in contrast to the

[7] We write the slack variables as the *margin slack vector* $\boldsymbol{\xi} = \boldsymbol{\xi}(h, S, \gamma) = [\xi_1, \cdots, \xi_N]$

optimal separating hyperplane, where we have set $R_{\text{emp}} = 0$, the generalised optimal separating hyperplane is found as a trade off between the empirical risk, measured by $F_\sigma(\boldsymbol{\xi})$, and the capacity of the classifier, measured by the norm of its weight vector $\|\mathbf{w}\|$. Its parameters $(\mathbf{w}^*, b^*)$ are the solution of the following quadratic programme.

**Proposition 8 (Generalised Optimal Separating Hyperplane).** *Suppose we are given the training set $S = ((\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N))$, $\mathbf{x}_n \in \mathbb{R}^{n_x}$, $y_n \in \{-1, +1\}$. Then, for some constant $C$, the hyperplane $(\mathbf{w}, b)$ that solves the quadratic programme*

$$\begin{aligned} \min_{\boldsymbol{\xi}\in\mathbb{R}^n_+, \mathbf{w}\in\mathbb{R}^{n_x}, b\in\mathbb{R}} \quad & \langle \mathbf{w}, \mathbf{w} \rangle + C F_\sigma(\boldsymbol{\xi}), \\ \text{subject to} \quad & y_n \left( \langle \mathbf{w}, \mathbf{x}_n \rangle + b \right) \geq 1 - \xi_n \\ & \xi_n \geq 0, \qquad n = 1, \cdots, N, \end{aligned}$$

*is the generalised optimal separating hyperplane with geometric margin $\gamma = \frac{1}{\|\mathbf{w}\|}$ with respect to the reduced training set $S'$, where margin errors are measured by $\xi_n$ [13].*

The parameter $C$ defines the trade off between empirical error and the capacity of the classifier. In practice, the optimal value of $C$ is assessed by varying it through a wide range of values and using some validation technique (Section 2.6).

As the strategy is only efficient for $\sigma = 1$ and $\sigma = 2$, it does not directly minimise bound (2.8), since for these values $F_\sigma(\boldsymbol{\xi})$ is only an upper bound on $R_{\text{emp}}$. Cristianini and Shawe-Taylor derived the following bounds on the risk of linear indicator functions.

**Theorem 9.** *Consider thresholding real-valued linear functions $\mathcal{H}$ with unit weight vector on an inner product space $X$ and fix $\gamma \in \mathbb{R}^+$. There is a constant $c$, such that for any probability distribution $P(\mathbf{x}, y)$ on $X \times \{-1, 1\}$ with support in a ball of radius $R$ around the origin, with probability $1 - \delta$ over $N$ random samples $S$, any hypothesis $h \in \mathcal{H}$ has error no more than*

$$\text{err}_{P(\mathbf{x},y)}(h) \leq \frac{c}{N} \left( \frac{R^2 + \|\boldsymbol{\xi}\|_2^2}{\gamma^2} \log^2 N + \log \frac{1}{\delta} \right) \quad \text{(2-norm)},$$

$$\text{err}_{P(\mathbf{x},y)}(h) \leq \frac{c}{N} \left( \frac{R^2 + \|\boldsymbol{\xi}\|_1^2 \log\left(\frac{1}{\gamma}\right)}{\gamma^2} \log^2 N + \log \frac{1}{\delta} \right) \quad \text{(1-norm)},$$

*where $\boldsymbol{\xi} = \boldsymbol{\xi}(h, S, \gamma)$ is the margin slack vector with respect to $h$ and $\gamma$ [13].*

Minimising these bounds leads to the same optimisation problems as in Proposition 8. Therefore, for an optimal choice of $C$, the solutions of these optimisation problems with the 1-norm and 2-norm empirical risk measure $F_\sigma(\boldsymbol{\xi})$ minimise the bounds in Theorem 9, respectively.

Notice that the support vector approach to finding the (generalised) optimal separating hyperplanes is not completely consistent with the SRM principle of Definition 3. The SRM principle requires the structure of sets of functions to be fixed *a priori*. In the support vector approach, the structure is defined through the margin *after* the data has been seen. This gap of theory is resolved with the so-called *data dependent structural risk minimisation* theory and the *luckiness* framework [62] [76].

### 2.2.4 The Support Vector Algorithms for Classification

The support vector approach to selecting the classifier with smallest generalisation error reduces to a quadratic programme (Proposition 7, 8). The solution to this optimisation problem is characterised by Lagrangian theory (Section 2.4). It allows for a dual form, which constitutes the support vector algorithms.

We recommend readers that are not familiar with Lagrangian theory to consult Section 2.4 before continuing with the present exposition.

#### The Algorithm for Hard Margin Classification

The solution of the optimisation problem (7) is given by the saddle point of the primal Lagrangian

$$L_P(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\langle \mathbf{w}, \mathbf{w}\rangle - \sum_{n=1}^{N} \alpha_n \left(y_n \left(\langle \mathbf{w}, \mathbf{x}_n\rangle + b\right) - 1\right), \; \alpha_n \geq 0, \tag{2.9}$$

where $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_N]$ are the Lagrange multipliers. It has to be minimised with respect to $\mathbf{w}$ and $b$, under the constraints $\alpha_n \geq 0$. By differentiating with respect to $\mathbf{w}$ and $b$, imposing stationarity,

$$\begin{aligned} \frac{\partial L_P(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{n=1}^{N} y_n \alpha_n \mathbf{x}_n = \mathbf{0} && \Rightarrow \mathbf{w} = \sum_{n=1}^{N} y_n \alpha_n \mathbf{x}_n, \\ \frac{\partial L_P(\mathbf{w}, b, \boldsymbol{\alpha})}{\partial b} &= \sum_{n=1}^{N} y_n \alpha_n = 0 && \Rightarrow 0 = \sum_{n=1}^{N} y_n \alpha_n. \end{aligned}$$

and substituting back into the primal (2.9) we obtain the dual objective function

$$W(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} y_m y_n \alpha_m \alpha_n \langle \mathbf{x}_m, \mathbf{x}_n\rangle.$$

that has to be maximised with respect to $\alpha_n \geq 0$.

**Proposition 10 (Hard Margin SVM).** *Consider a linearly separable training set* $S = ((\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N))$ *and suppose the parameters* $\boldsymbol{\alpha}^*$ *solve the following quadratic optimisation problem:*

$$\begin{aligned} \max_{\boldsymbol{\alpha} \in \mathbb{R}_+^{n_x}} \quad & W(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} y_m y_n \alpha_m \alpha_n \langle \mathbf{x}_m, \mathbf{x}_n\rangle, \\ \text{subject to} \quad & \sum_{n=1}^{N} y_n \alpha_n = 0, \quad \alpha_n \geq 0; \quad n = 1, \cdots, N. \end{aligned}$$

*Then the weight vector* $\mathbf{w}^* = \sum_{n=1}^{N} y_n \alpha_n^* \mathbf{x}_n$ *solves the optimisation problem (7) and realises the maximal margin hyperplane with geometric margin* $\gamma = \frac{1}{\|\mathbf{w}^*\|_2}$ *[13].*

The optimal solution $\boldsymbol{\alpha}^*$, $(\mathbf{w}^*, b^*)$ must satisfy the *Karush Kuhn Tucker (KKT) complementary conditions* (Theorem 21, Section 2.4),

$$\alpha_n^* \left( y_n(\langle \mathbf{w}, \mathbf{x}_n \rangle + b^+) - 1 \right) = 0, \qquad n = 1, \cdots, N.$$

Therefore, only for vectors $\mathbf{x}_n$ for which the functional margin is 1 and which thus lie closest to the hyperplane, the corresponding multipliers $\alpha_n$ are non-zero. These vectors are called the *support vectors*. It is only these vectors that appear in the final expression of the weight vector $\mathbf{w}$ and the final hypothesis[8]

$$h(\mathbf{x}, \boldsymbol{\alpha}^*, b^*) = \operatorname{sign}\left( \sum_{n \in \mathcal{I}_{\text{sv}}} y_n \alpha_n^* \langle \mathbf{x}_n, \mathbf{x} \rangle + b^* \right). \tag{2.10}$$

The variable $b$ does not appear in the dual problem, and its optimal value $b^*$ can be found by application of the KKT complementary conditions.

### The Algorithms for Soft Margin Classification

The solution of the optimisation problem (8) for $F_1(\boldsymbol{\xi})$ and $F_2(\boldsymbol{\xi})$, respectively, are given by the saddle points of the primal Lagrangians

$$\text{1-norm: } L_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle + C \sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} \alpha_n \left( y_n \left( \langle \mathbf{w}, \mathbf{x}_n \rangle + b \right) - 1 + \xi_n \right) - \sum_{n=1}^{N} r_n \xi_n, \tag{2.11}$$

$$\text{2-norm: } L_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}) = \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle + \frac{C}{2} \sum_{n=1}^{N} \xi_n^2 - \sum_{n=1}^{N} \alpha_n \left( y_n \left( \langle \mathbf{w}, \mathbf{x}_n \rangle + b \right) - 1 + \xi_n \right), \tag{2.12}$$

$$\alpha_n, r_n \geq 0$$

which have to be minimised with respect to $\mathbf{w}$, $\boldsymbol{\xi}$ and b. By differentiating with respect to $\mathbf{w}$ and $b$, imposing stationarity,

| 1-norm soft margin | 2-norm soft margin |
|---|---|
| $\dfrac{\partial L_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N} y_n \alpha_n \mathbf{x}_n = \mathbf{0}$ | $\dfrac{\partial L_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha})}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N} y_n \alpha_n \mathbf{x}_n = \mathbf{0},$ |
| $\dfrac{\partial L_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r})}{\partial \xi_n} = C - \alpha_n - r_n = \mathbf{0}$ | $\dfrac{\partial L_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha})}{\partial \boldsymbol{\xi}} = C\boldsymbol{\xi} - \boldsymbol{\alpha} = \mathbf{0},$ |
| $\dfrac{\partial L_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r})}{\partial b} = \sum_{n=1}^{N} y_n \alpha_n = 0$ | $\dfrac{\partial L_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha})}{\partial b} = \sum_{n=1}^{N} y_n \alpha_n = 0,$ |

[8] Throughout this work, $\mathcal{I}_{\text{sv}}$ denotes the set of indices of the support vectors, i. e., the subset of indices $\mathcal{I}_{\text{sv}} \subseteq \{1, \cdots, N\}$ for which $\alpha_n \neq 0$.

and substituting back into the primals (2.11) and (2.12), we obtain the dual objective functions

$$\text{1-norm soft margin:} \quad W(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} y_m y_n \alpha_m \alpha_n \langle \mathbf{x}_m, \mathbf{x}_n \rangle,$$

$$\text{2-norm soft margin:} \quad W(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} y_m y_n \alpha_m \alpha_n \left( \langle \mathbf{x}_m, \mathbf{x}_n \rangle + \frac{1}{C} \delta_{mn} \right),$$

that have to be maximised with respect to the dual variables $\alpha_n$. The optimal solutions $\boldsymbol{\alpha}^*$, $(\mathbf{w}^*, b^*)$ must satisfy the KKT complementary conditions (Theorem 21, Section 2.4)

$$\begin{aligned} \text{1-norm soft margin:} \quad \alpha_n \left( y_n \left( \langle \mathbf{x}_n, \mathbf{w} \rangle + b \right) - 1 + \xi_n \right) &= 0, \quad n = 1, \cdots, N, \\ \xi_n (\alpha_n - C) &= 0, \quad n = 1, \cdots, N, \\ \text{2-norm soft margin:} \quad \alpha_n \left( y_n \left( \langle \mathbf{x}_n, \mathbf{w} \rangle + b \right) - 1 + \xi_n \right) &= 0, \quad n = 1, \cdots, N, \end{aligned}$$

implying that all samples $(\mathbf{x}_n, y_n)$ for which $y_n(\langle \mathbf{w}, \mathbf{x}_n \rangle + b) > 1$ must have corresponding multipliers $\alpha_n = 0$, since the slack variables $\xi_n$ cannot be negative. Thus, the solution has again a sparse representation (2.10) in terms of support vectors.
The optimal $b^*$ can be found by application of the KKT complementary conditions.

**Proposition 11 (1-norm Soft Margin SVC).** *Consider classifying a training sample $S = ((\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N))$ and suppose the parameters $\boldsymbol{\alpha}^*$ solve the following quadratic optimisation problem:*

$$\begin{aligned} \max_{\alpha_n \in [0,C]} \quad & W(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} y_m y_n \alpha_m \alpha_n \langle \mathbf{x}_m, \mathbf{x}_n \rangle \\ \text{subject to} \quad & \sum_{n=1}^{N} y_n \alpha_n = 0, \quad C \geq \alpha_n \geq 0; \quad n = 1, \cdots, N. \end{aligned}$$

*Then the weight vector $\mathbf{w}^* = \sum_{n=1}^{N} y_n \alpha_n^* \mathbf{x}_n$ realises the generalised optimal separating hyperplane, where the slack variables are defined relative to the geometric margin $\gamma = \left( \sum_{m,n \in \mathcal{I}_{\text{sv}}} y_m y_n \alpha_m^* \alpha_n^* \langle \mathbf{x}_m, \mathbf{x}_n \rangle \right)^{-\frac{1}{2}}$ [13].*

**Proposition 12 (2-norm Soft Margin SVC).** *Consider classifying a training sample $S = ((\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N))$, and suppose the parameters $\boldsymbol{\alpha}^*$ solve the following quadratic optimisation problem:*

$$\begin{aligned} \max_{\alpha_n \in \mathbb{R}^+} \quad & W(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} y_m y_n \alpha_m \alpha_n \left( \langle \mathbf{x}_m, \mathbf{x}_n \rangle + \frac{1}{C} \delta_{mn} \right) \\ \text{subject to} \quad & \sum_{n=1}^{N} y_n \alpha_n = 0, \quad \alpha_n \geq 0; \quad n = 1, \cdots, N. \end{aligned}$$

*Then the weight vector $\mathbf{w}^* = \sum_{n=1}^{N} y_n \alpha_n^* \mathbf{x}_n$ realises the generalised optimal separating hyperplane, where the slack variables are defined relative to the geometric margin $\gamma = \left( \sum_{n \in \mathcal{I}_{\text{sv}}} \alpha_n^* - \frac{1}{C} \langle \boldsymbol{\alpha}^*, \boldsymbol{\alpha}^* \rangle \right)^{-\frac{1}{2}}$ [13].*

### 2.2.5 Notes on the Support Vector Classification Machines

The SRM solution to finding a linear classifier is the hyperplane that has maximal margin on the (reduced) training set. These hyperplanes have maximal distance from the training vectors $\mathbf{x}_n$ that produce no training error. Therefore, slightly perturbing $\mathbf{x}_n$ will not result in an error on this vector as long as we do not move at least $\gamma$ units in the direction of the hyperplane. The (generalised) optimal separating hyperplanes are the most robust possible hyperplanes with respect to changes in the training samples, since they have maximal margin[9]. They thus minimise the probability of classification error on unseen samples, and therefore minimise the risk.

We note the following key properties of the support vector classifiers:

1. The problem of finding the classifier that is optimal in terms of generalisation is formulated as a quadratic programme. This quadratic programme can be solved efficiently. The solution is unique.

2. The weight vector $\mathbf{w}^*$ of the final hypothesis is a linear combination of only a fraction of the training vectors, termed support vectors. The solution has a sparse representation.

3. In both the training algorithms (Proposition 10, 11, 12) and in the final hypotheses of from (2.10), the data enters only inside of inner products $\langle \cdot, \cdot \rangle$. This allows for a generalisation of the algorithms to a nonlinear hypothesis space by simply replacing all inner products by a kernel (Section 2.5).

We will see in the next section that SVMs for regression estimation as well possess all of these properties. The regressor with minimal risk is the solution to a convex quadratic optimisation problem, and has a sparse representation in terms of support vectors. An extension of the linear support vector regressors to nonlinear hypothesis spaces through the use of kernels is possible. The interpretation of the solution as the hyperplane having maximal margin on the training set will, however, be lost. The SVR will choose the *flattest* hypothesis.

Finally, we note that the only difference between the maximal margin classifier algorithm and the soft margin algorithms is that

1. in the 1-norm case, the dual variables $\alpha_n$ are subject to an additional constraint $\alpha_n \leq C$, and

2. in the 2-norm case, $\frac{1}{C}$ is added to the diagonal of the Gram matrix $\boldsymbol{G}$ with entries $G_{mn} = \langle \mathbf{x}_m, \mathbf{x}_n \rangle$.

[9]The generalised optimal separating hyperplane has maximal margin on the reduced training set whose samples do not produce training errors and has at the same time small training error.

## 2.3 Support Vector Regression (SVR)

### 2.3.1 The Regression Problem

Suppose we are given a training set

$$S = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}, \quad \mathbf{x}_n \in X \subseteq \mathbb{R}^{n_x}, \quad y_n \in Y \subseteq \mathbb{R}, \tag{2.13}$$

with $N$ samples $(\mathbf{x}_n, y_n)$ that are drawn i.i.d. according to some unknown but fixed probability distribution $P(\mathbf{x}, y)$. Let the hypothesis space be the class of linear functions in the $n_x$-dimensional space (hyperplanes),

$$\mathcal{H} = \{h | h(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b; \quad \mathbf{w}, \mathbf{x} \in \mathbb{R}^{n_x}; \quad b \in \mathbb{R}\}, \tag{2.14}$$

Then the learning problem is to select the hypothesis that minimises the risk (2.2),

$$R(h) = \int L(y, h(\mathbf{x})) dP(\mathbf{x}, y),$$

based on the training set $S$, where $L(y, h(\mathbf{x}))$ is a suitably chosen loss function. The loss function indicates how differences between $y$ and $h(\mathbf{x})$ (residuals) are penalised.
As discussed in Section 2.1, the risk cannot be evaluated directly, but we can try to select a hypothesis that minimises an upper bound on the risk by means of the SRM induction principle.

The choice of hyperplanes as the hypothesis space $\mathcal{H}$ may seem limiting. Similar to the algorithms for support vector classification, the algorithms derived for regression estimation will, however, allow for a generalisation to a rich nonlinear hypothesis space by simply replacing the inner products through which the data enter the algorithms by a kernel function (Section 2.5).

### 2.3.2 The Loss Function: $\epsilon$-Precision Regression

We need to choose a loss function that is appropriate for the regression problem, for which the labels $y$ are no longer discrete but chosen from $Y \subseteq \mathbb{R}$. Vapnik proposed the $\epsilon$-insensitive loss functions for support vector regression [67].

**Definition 13 ($\epsilon$-Insensitive Loss Function).** *The linear $\epsilon$-insensitive loss function $L_1^{(\epsilon)}(y, h(\mathbf{x}))$ is defined by*

$$L_1^{(\epsilon)}(y, h(\mathbf{x})) = |y - h(\mathbf{x})|_\epsilon \stackrel{\triangle}{=} \max(0, |y - h(\mathbf{x})| - \epsilon),$$

*where $h$ is a real-valued function on a domain $X$, $\mathbf{x} \in X$ and $y \in \mathbb{R}$. Similarly, the quadratic $\epsilon$-insensitive loss function is given by*

$$L_2^{(\epsilon)}(y, h(\mathbf{x})) = |y - h(\mathbf{x})|_\epsilon^2 \stackrel{\triangle}{=} \max\left(0, (|y - h(\mathbf{x})| - \epsilon)^2\right).$$

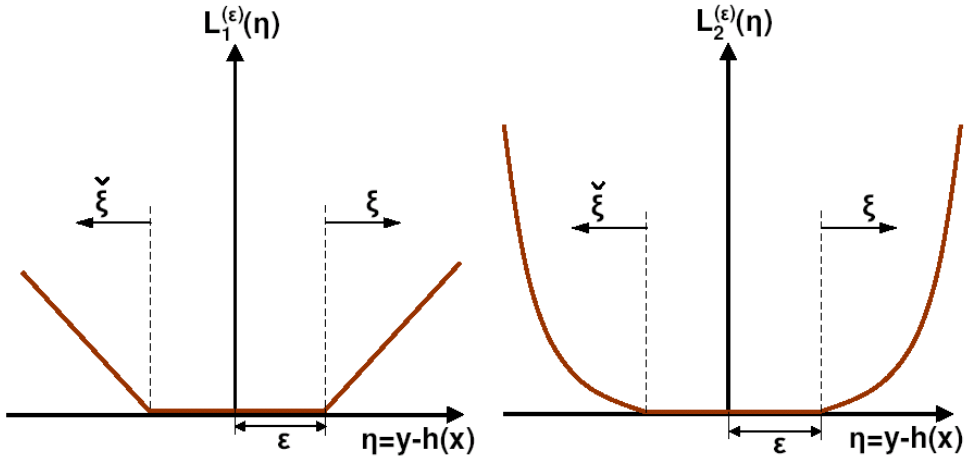


Figure 2.6: *Linear (left) and quadratic (right) $\epsilon$-insensitive loss function. Only deviations larger than $\epsilon$ count as training errors. These deviations are captured by slack variables $\xi$, $\check{\xi}$.*

In other words, we do not count a training sample as an error as long as its residual is below some given value $\epsilon$ (Figure 2.6).
Loss functions having an $\epsilon$-insensitive zone ensure that the solution hypothesis has a sparse representation in terms of support vectors. Thus, the $\epsilon$-insensitive zone can somehow be seen as an analogue of the margin in classification [52]. This analogy is, however, limited, since $\epsilon$ is fixed before training and not optimised through the training algorithm. The size of $\epsilon$ does not directly enter into bounds on the risk and does not control the capacity of the hypotheses.

The loss functions $L_1^{(\epsilon)}$ and $L_2^{(\epsilon)}$ can be seen as generalisations of the Laplacian and Gaussian loss. For a more general discussion on loss functions, we refer to section 2.3.10.

We capture the amount by which a training sample fails to have training target accuracy $\epsilon$ by slack variables

$$\xi((\mathbf{x}_n, y_n), h, \epsilon) = \xi_n \triangleq \max(0, y_n - h(\mathbf{x}_n) - \epsilon),$$
$$\check{\xi}((\mathbf{x}_n, y_n), h, \epsilon) = \check{\xi}_n \triangleq \max(0, h(\mathbf{x}_n) - y_n - \epsilon).$$

Here, $\xi$ and $\check{\xi}$ correspond to training samples that happen to lie above and below the hypothesis, respectively[10] (Figures 2.6 and 2.7).

### 2.3.3 Constructing the Learning Machine

Historically, SVR have been introduced by translating the optimisation problem in Proposition 8 for the regression problem [12]. This was done by replacing the empirical risk measured by $F_\sigma(\boldsymbol{\xi})$ with an empirical risk measured by the $\epsilon$-insensitive loss functions $L_1^{(\epsilon)}$

[10] We write the slack variables as the *slack vector* $\boldsymbol{\xi} = \boldsymbol{\xi}(h, S, \gamma) = [\xi_1 + \check{\xi}_1, \cdots, \xi_N + \check{\xi}_N]$.

and $L_2^{(\epsilon)}$. The resulting optimisation problem then takes the form

$$\begin{aligned} \min_{\mathbf{w}\in\mathbb{R}^{n_x},b\in\mathbb{R}} \quad & \langle \mathbf{w},\mathbf{w}\rangle + C\sum_{n=1}^{N}(\xi^{(2)} + \check{\xi}^{(2)}), \\ \text{subject to} \quad & (\langle \mathbf{w},\mathbf{x}_n\rangle + b) - y_n \leq \epsilon + \xi_n, \\ & y_n - (\langle \mathbf{w},\mathbf{x}_n\rangle + b) \leq \epsilon + \check{\xi}_n, \\ & \xi_n, \check{\xi}_n \geq 0, \qquad\qquad n = 1,\cdots,N, \end{aligned}$$

At the time of their introduction, this analogy has not been theoretically justified, since the bound that Proposition 8 is derived from cannot be applied for these loss functions. Later on, bounds on the risk that hold for $\epsilon$-insensitive loss functions have been found, and minimising these bounds leads to the same optimisation problem as stated above. We will investigate such a bound in the following subsection.

### 2.3.4 A Bound on the Generalisation of a Linear Regression Estimator

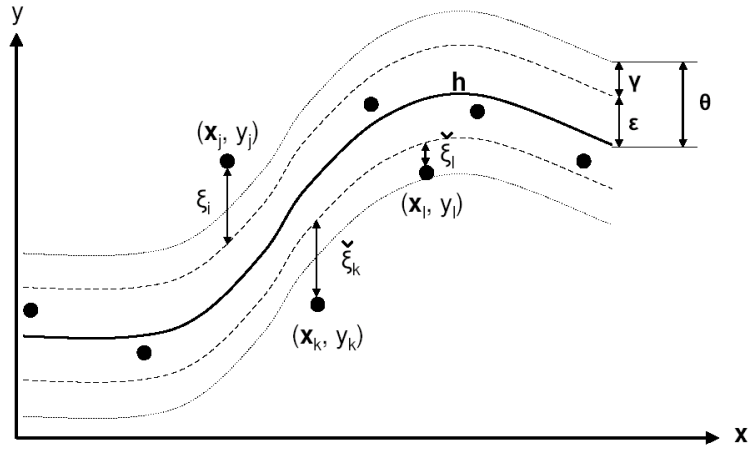


Figure 2.7: *$\epsilon$-precision regression. Samples that happen to lie outside of a band $\pm\epsilon$ around the hypothesis count as training errors, samples outside a band $\pm\theta$ count as well as test errors. The sample $(\mathbf{x}_l, y_l)$ counts as a training error, but not as a test error. The samples $(\mathbf{x}_j, y_j)$ and $(\mathbf{x}_k, y_k)$ are training and test errors.*

We aim to obtain a bound on the probability that a randomly drawn test sample has accuracy less than $\theta$, i.e., a bound on the risk $R(h) = \Pr(|y_n - h(\mathbf{x}_n)| \geq \theta) = \mathrm{err}_{P(\mathbf{x},y)}(h)$. For doing so, we introduce a margin $\gamma$ in the training regression accuracy, such that a training sample counts as an error if $|y_n - h(\mathbf{x}_n)| \geq \theta - \gamma = \epsilon$. Consequently, in *training*, a sample counts as a mistake if it is outside a band of size $\pm\epsilon = \pm(\theta - \gamma)$ around $h(\mathbf{x})$, and in *testing*, a sample counts as a mistake if it is outside a band of size $\pm\theta$ (Figure 2.7).

For this situation, the following theorem can be derived. It provides bounds on the risk of a hypothesis, i.e. on the probability that a test sample that is randomly drawn from $P(\mathbf{x}, y)$ will be more than $\theta$ units away from the hypothesis.

**Theorem 14.** *Consider performing regression with linear functions $h \in \mathcal{H}$ on an inner product space $X$ and fix $\gamma \leq \theta \in \mathbb{R}^+$. There is a constant $c$, such that for any probability distribution $P(\mathbf{x}, y)$ on $X \times \mathbb{R}$ with support in a ball of radius $R$ around the origin, with probability $1-\delta$ over $N$ random samples $S$, the probability that a hypothesis $(\mathbf{w}, b) \in \mathcal{H}$ has output more than $\theta$ away from its true value is bounded by*

$$\mathrm{err}_{P(\mathbf{x},y)}(h) \leq \frac{c}{N}\left(\frac{\|\mathbf{w}\|_2^2 R^2 + \|\boldsymbol{\xi}\|_2^2}{\gamma^2}\log^2 N + \log\frac{1}{\delta}\right) \quad \text{(2-norm)},$$

$$\mathrm{err}_{P(\mathbf{x},y)}(h) \leq \frac{c}{N}\left(\frac{\|\mathbf{w}\|_2^2 R^2 + \|\boldsymbol{\xi}\|_1^2 \log\left(\frac{1}{\gamma}\right)}{\gamma^2}\log^2 N + \log\frac{1}{\delta}\right) \quad \text{(1-norm)},$$

*where $\boldsymbol{\xi} = \boldsymbol{\xi}(\mathbf{w}, S, \theta-\gamma)$ is the slack vector with respect to $\mathbf{w}$, $\theta$ and $\gamma$ [13].*

Theorem 14 applies directly to linear and quadratic $\epsilon$-insensitive loss functions with $\epsilon = \theta - \gamma$. Note that similar bounds can be derived for other loss functions than the linear and quadratic $\epsilon$-insensitive loss [56].

**Motivation of Theorem 14 through Theorem 9.** We try to establish a link between Theorem 9, bounding the risk of classifiers, and Theorem 14, bounding the risk of regressors. Let $h$ be from the function class defined by (2.14), and let $\mathbf{c}_n = y_n - h(\mathbf{x}_n)$, where $(\mathbf{x}_n, y_n)$ are the samples from the training set (2.13).
Now consider the following thresholded linear functions as classifiers:

$$h_+^c(\mathbf{c}_n) = h_+^c(y_n - h(\mathbf{x}_n)) \begin{cases} +1 & \mathbf{c}_n = y_n - h(\mathbf{x}_n) - \theta \leq 0 \\ -1 & \text{otherwise,} \end{cases}$$

$$h_-^c(\mathbf{c}_n) = h_+^c(y_n - h(\mathbf{x}_n)) \begin{cases} +1 & \mathbf{c}_n = y_n - h(\mathbf{x}_n) + \theta \geq 0 \\ -1 & \text{otherwise.} \end{cases}$$

If we label the attribute vectors $\mathbf{c}_n$ with $y^c$ such that

$$\mathcal{D} : \{P(\mathbf{c}), \quad P(y^c = +1|\mathbf{c}) = 1, \quad P(y^c = -1|\mathbf{c}) = 0\},$$

then $\Pr(|y_n - h(\mathbf{x}_n)| \geq \theta)$ is

$$Pr(|y_n - h(\mathbf{x}_n)| \geq \theta) = \mathrm{err}_{\mathcal{D}}(h_+^C) + \mathrm{err}_{\mathcal{D}}(h_-^C).$$

Therefore, the risk for regression can be expressed in terms of the risk of these classifiers, since a vector $\mathbf{c} = y_n - h(\mathbf{x}_n)$ produces a classification error if and only if $|y_n - h(\mathbf{x}_n)| > \theta$.

The probability of classification error for $h_+^c$ and $h_-^c$ with unit weight vector is given by (9). In the case of regression, it makes no longer sense to fix $\|\mathbf{w}\|_2^2 = 1$, since rescaling the weight vector changes the functionality of the regressor, and therefore the term $R^2$ has to change to[11] $R^2\|\mathbf{w}\|_2^2$. The terms $\|\xi\|_2^2$ and $\|\xi\|_1^2$ do not change as they contain the slack variables scaled by $\|\mathbf{w}\|_2^2$. The left and center column of Figure 2.8 illustrate the

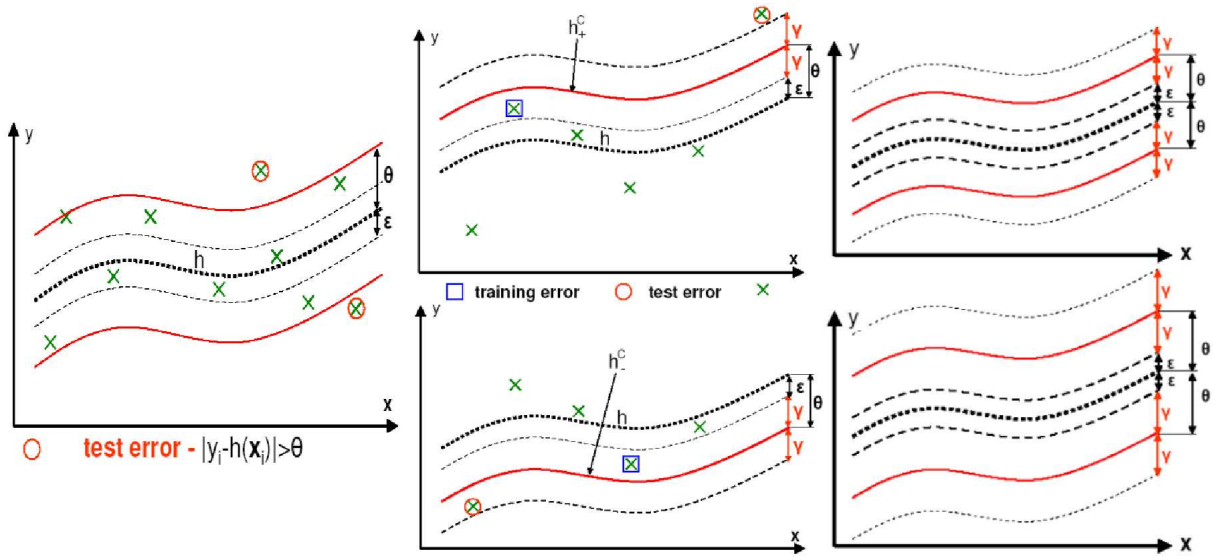


Figure 2.8: *Illustration motivation of Theorem 14 through Theorem 9. Samples with precision less than $\epsilon$ and $\theta$ count as training errors and test errors, respectively (left). The classifiers $h^c_+$ and $h^c_-$ for the cases of being above and below target precision. The probability of classification error equals the probability of $|y_n - h(\mathbf{x}_n)| > \theta$ (center). The margin $\gamma$ of the classifiers does not have an influence on the structure of the bounds in Theorem 14 (right).*

connection between Theorem 14 and Theorem 9.$\Diamond$

Minimising the bounds in Theorem 14 over the class of linear functions (2.14) is equivalent to minimising its numerator, since it is the only part we can influence by choosing the hypothesis. Thus, when the width of the insensitive region $\epsilon = \theta - \gamma$ is fixed, we minimise the bounds for all values of $\gamma$ and $\theta$, if we ignore the log-factor in the 1-norm case.

### The Support Vector Regressors

Minimising the bounds on the risk given in Theorem 14 is equivalent to jointly minimising the norm of the weight vector $\|\mathbf{w}\|_2^2$ and some norm of the slack vector $\boldsymbol{\xi}$. The theorem proposes optimal values for the trade off between capacity (measured by $\|\mathbf{w}\|_2^2$) and training error (measured by the 1-norm or 2-norm of the slack variable vector). In the support vector approach to minimising these bounds, a parameter $C$ is introduced to trade off between capacity and training error of the hypothesis, and the optimal value of $C$ is assessed via some validation technique. As the parameter $C$ runs through a range of values, the norm $\|\mathbf{w}\|$ varies smoothly through a corresponding range. Hence, in a particular problem, choosing a value for $C$ corresponds to choosing a value for $\|\mathbf{w}\|$ and then minimising the 1-norm or 2-norm of the slack vector. Since there is a value of $C$ that

[11] In deriving (9), one bounds the classification error via the covering numbers of $\mathcal{H}$, and the covering numbers via the fat shattering dimension, which is bounded by $fat_{\mathcal{H}}(\gamma) \leq \left(\frac{\|\mathbf{w}\| \cdot R}{\gamma}\right)^2$ [13] [55].

corresponds to the optimal choice of $\|\mathbf{w}\|$, this value of $C$ will give the optimal bound as it will correspond to find the minimum of the 1-norm or 2-norm of the slack vector with the given value of[12] $\|\mathbf{w}\|$ [13].

Notice that the support vector strategy is not completely consistent with the definition of the SRM principle. The SRM principle requires the structure on the hypothesis space to be fixed *a priori*, and to select the hypothesis having lowest empirical risk within the hypothesis subclass for which the bound on the risk is minimal. In the support vector approach, the hypothesis minimising the bound on the risk is found by jointly minimising the norm of the weight vector and the norm of the slack vector, such that the structure is defined by variation of the norm of the weight vector *after* the data has been seen. This gap of theory is resolved with the so-called *data dependent structural risk minimisation* theory and the *luckiness* framework (for details, see e.g. [62] and [76]).

### 2.3.5 Support Vector Regression with Linear $\epsilon$-Insensitive Loss

Theorem 14 suggests to minimise the expression

$$R^2\|\mathbf{w}\|_2^2 + \sum_{n=1}^{N} L_1^{(\epsilon)}(y_n, \langle \mathbf{w}, \mathbf{x}_n \rangle + b) \log\left(\frac{1}{\gamma}\right).$$

The support vector approach is ignoring the log-factor and minimises

$$\frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{n=1}^{N} L_1^{(\epsilon)}(y_n, \langle \mathbf{w}, \mathbf{x}_n \rangle + b),$$

by varying $C$ through a range of values as described above to arrive at the optimal solution. This can be reformulated as a quadratic programme,

$$\min_{\mathbf{w}\in\mathbb{R}^{n_x}, b\in\mathbb{R}} \quad \|\mathbf{w}\|_2^2 + C\sum_{n=1}^{N}(\xi_n + \check{\xi}_n), \tag{2.15}$$

$$\text{subject to} \quad (\langle \mathbf{w}, \mathbf{x}_n \rangle + b) - y_n \leq \epsilon + \check{\xi}_n, \tag{2.16}$$

$$y_n - (\langle \mathbf{w}, \mathbf{x}_n \rangle + b) \leq \epsilon + \xi_n, \tag{2.17}$$

$$\xi_n, \check{\xi}_n \geq 0, \qquad n = 1, \cdots, N, \tag{2.18}$$

where the two slack variables $\xi_n, \check{\xi}_n$ are for the cases of exceeding the target value by $\epsilon$, and for being more than $\epsilon$ below the target value, respectively, as illustrated in Figures 2.6 and 2.7.

[12] An alternative point of view is to consider the trade off parameter $C$ as an additional linear scaling in the loss functions, $L'(y, h(\mathbf{x})) = CL(y, h(\mathbf{x}))$. Then, the value of $C$ corresponding to the optimal choice of $\|\mathbf{w}\|$ corresponds to a particularly scaled loss function.

The primal Lagrangian of this problem is

$$L_P(\mathbf{w}, b, \boldsymbol{\xi}, \check{\boldsymbol{\xi}}, \boldsymbol{\alpha}, \check{\boldsymbol{\alpha}}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{n=1}^{N}(\xi_n + \check{\xi}_n) - \sum_{n=1}^{N}\check{\alpha}_n(\epsilon + \check{\xi}_n + y_n - \langle \mathbf{w}, \mathbf{x}_n\rangle - b) - \tag{2.19}$$

$$- \sum_{n=1}^{N}\alpha_n(\epsilon + \xi_n - y_n + \langle \mathbf{w}, \mathbf{x}_n\rangle + b) - \sum_{n=1}^{N}(r_n\xi_n + \check{r}_n\check{\xi}_n) \tag{2.20}$$

Since the solution of this problem has to be a saddle point (Chapter 2.4), it follows that the partial derivatives of $L_P(\mathbf{w}, b, \boldsymbol{\xi}, \check{\boldsymbol{\xi}}, \boldsymbol{\alpha}, \check{\boldsymbol{\alpha}})$ with respect to the primal variables $\mathbf{w}$, $b$, $\xi_n$, $\check{\xi}_n$ have to vanish,

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N}(\alpha_n - \check{\alpha}_n)\mathbf{x}_n = \mathbf{0} \qquad \Rightarrow \mathbf{w} = \sum_{n=1}^{N}(\alpha_n - \check{\alpha}_n)\mathbf{x}_n \tag{2.21}$$

$$\frac{\partial L_P}{\partial \xi_n} = C - \alpha_n - r_n = 0 \qquad \Rightarrow r_n = C - \alpha_n \tag{2.22}$$

$$\frac{\partial L_P}{\partial \check{\xi}_n} = C - \check{\alpha}_n - \check{r}_n = 0 \qquad \Rightarrow \check{r}_n = C - \check{\alpha}_n \tag{2.23}$$

$$\frac{\partial L_P}{\partial b} = \sum_{n=1}^{N}(\check{\alpha}_n - \alpha_n) = 0. \tag{2.24}$$

Substituting back into (2.19), we obtain the dual objective function

$$\begin{aligned} W(\boldsymbol{\alpha}, \check{\boldsymbol{\alpha}}) = {} & \frac{1}{2}\sum_{m,n=1}^{N}(\alpha_m - \check{\alpha}_m)(\alpha_n - \check{\alpha}_n)\langle \mathbf{x}_m, \mathbf{x}_n\rangle + C\sum_{n=1}^{N}(\xi_n + \check{\xi}_n) - \epsilon\sum_{n=1}^{N}(\alpha_n + \check{\alpha}_n) \\ & - \sum_{n=1}^{N}(\alpha_n\xi_n + \check{\alpha}_n\check{\xi}_n) + \sum_{n=1}^{N} y_n(\alpha_n - \check{\alpha}_n) - \sum_{m,n=1}^{N}(\alpha_m - \check{\alpha}_m)(\alpha_n - \check{\alpha}_n)\langle \mathbf{x}_m, \mathbf{x}_n\rangle \\ & - b\sum_{n=1}^{N}(\alpha_n - \check{\alpha}_n) - C\sum_{n=1}^{N}(\xi_n + \check{\xi}_n) + \sum_{n=1}^{N}(\alpha_n\xi_n + \check{\alpha}_n\check{\xi}_n) = \\ = {} & -\frac{1}{2}\sum_{m,n=1}^{N}(\alpha_m - \check{\alpha}_m)(\alpha_n - \check{\alpha}_n)\langle \mathbf{x}_m, \mathbf{x}_n\rangle - \epsilon\sum_{n=1}^{N}(\alpha_n + \check{\alpha}_n) + \sum_{n=1}^{N} y_n(\alpha_n - \check{\alpha}_n) \end{aligned}$$

that has to be maximised with respect to the dual variables $\alpha_n, \check{\alpha}_n$. The dual optimisation problem is therefore

$$\begin{aligned} \max_{\boldsymbol{\alpha}, \check{\boldsymbol{\alpha}} \in [\mathbf{0}, \boldsymbol{C}]} \quad & -\frac{1}{2}\sum_{m,n=1}^{N}(\alpha_m - \check{\alpha}_m)(\alpha_n - \check{\alpha}_n)\langle \mathbf{x}_m, \mathbf{x}_n\rangle - \epsilon\sum_{n=1}^{N}(\alpha_n + \check{\alpha}_n) + \sum_{n=1}^{N} y_n(\alpha_n - \check{\alpha}_n) \\ \text{subject to} \quad & \sum_{n=1}^{N}(\check{\alpha}_n - \alpha_n) = 0; \quad \alpha_n,\, \check{\alpha}_n \in [0, C]. \end{aligned}$$

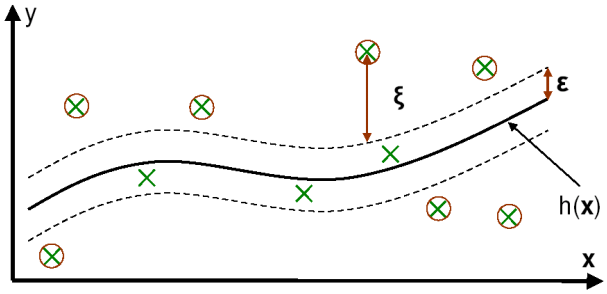


Figure 2.9: *Illustration of the KKT complementary conditions. Samples that happen to lie inside the $\epsilon$-insensitive region of the hypothesis $h(\mathbf{x})$ must have corresponding Lagrangian multipliers $\alpha, \check{\alpha} = 0$. Only samples lying on the border or outside of the $\epsilon$-insensitive region can have non-zero Lagrangian multipliers and contribute to the solution (support vectors - circled).*

This is still a quadratic programme. We note that $\xi_n$, $\alpha_n$, $\check{\xi}_n$, $\check{\alpha}_n$ must satisfy

$$\xi_n \check{\xi}_n = 0, \quad \alpha_n \check{\alpha}_n = 0,$$

as $\xi_n \check{\xi}_n \neq 0$ or $\alpha_n \check{\alpha}_n \neq 0$ would require non-zero slack variables in both directions. The KKT complementary conditions (Theorem 21, Section 2.4) corresponding to this optimisation problem are

$$\begin{aligned}
\check{\alpha}_n \left( \langle \mathbf{w}, \mathbf{x}_n \rangle + b - y_n - \epsilon - \check{\xi}_n \rangle \right) &= 0, \\
\alpha_n \left( y_n - \langle \mathbf{w}, \mathbf{x}_n \rangle - b - \epsilon - \xi_n \rangle \right) &= 0, \\
\xi_n \check{\xi}_n = 0, \qquad & \alpha_n \check{\alpha}_n = 0, \\
(\alpha_n - C)\xi_n = 0, \quad (\check{\alpha}_n - C)\check{\xi}_n = 0, \quad & n = 1, \cdots, N.
\end{aligned}$$

Therefore, only vectors $\mathbf{x}_n$ with $|h(\mathbf{x}_n) - y_n| \geq \epsilon$ can have non-zero multipliers $\alpha_n$ or $\check{\alpha}_n$. These vectors are the support vectors. If a strict inequality holds, $\alpha_n = C$ or $\check{\alpha}_n = C$. Samples lying on the border of the $\epsilon$-tube have corresponding multipliers in the range $[0, C]$, whereas samples lying inside of the $\epsilon$-tube must have corresponding multipliers with value zero, since the second factor in one of the first two KKT complementary conditions then is non-zero. As a consequence, the weight vector (2.21) of the solution has a sparse representation in terms of support vectors, and the final hypothesis has the form

$$h(\mathbf{x}) = \sum_{n \in \mathcal{I}_{\text{sv}}} (\alpha_n^* - \check{\alpha}_n^*) \langle \mathbf{x}_n, \mathbf{x} \rangle + b^*. \tag{2.25}$$

The effect of the KKT complementary conditions is illustrated in Figure 2.9. The value of $b^*$ can be found by applying them to a vector $(\mathbf{x}_n, y_n)$ with non-zero multiplier $\alpha_n$ or $\check{\alpha}_n$.

By substituting the expansion coefficients $\beta_n$ for our original multipliers $\alpha_n, \check{\alpha}_n$, $\boldsymbol{\beta} = \boldsymbol{\alpha} - \check{\boldsymbol{\alpha}}$, we can rewrite the dual problem in a form closely resembling the classification

case (Proposition 11),

$$\begin{aligned} \max_{\boldsymbol{\beta}\in[-\boldsymbol{C},\boldsymbol{C}]} \quad & -\frac{1}{2}\sum_{m,n=1}^{N}\beta_m\beta_n\langle\mathbf{x}_m,\mathbf{x}_n\rangle - \epsilon\sum_{n=1}^{N}|\beta_n| + \sum_{n=1}^{N}y_n\beta_n \\ \text{subject to} \quad & \sum_{n=1}^{N}\beta_n = 0, \quad \beta_n \in [-C,C], \quad n = 1,\cdots,N. \end{aligned}$$

For $y_n \in \{-1,+1\}$ and ${\beta'}_n = y_n\beta_n$, the similarity becomes even more obvious, with the only difference being that ${\beta'}_n$ is not constrained to be in a positive range any more, and that for non-zero $\epsilon$, an additional weight decay factor involving the dual parameters is introduced.

In the following proposition, we summarises the support vector algorithm for regression with linear $\epsilon$-insensitive loss. We have moved directly to a more general hypothesis space by replacing all inner products appearing in the algorithm with a kernel function $K(\cdot,\cdot)$ calculating the inner product in some feature space (Section 2.5).

**Proposition 15 (SVR with Linear $\epsilon$-Insensitive Loss).** *Suppose that we wish to perform regression on a training sample $S = ((\mathbf{x}_1,y_1),\cdots,(\mathbf{x}_N,y_N))$ and suppose the parameters $\boldsymbol{\beta}^*$ solve the following optimisation problem:*

$$\begin{aligned} \max_{\boldsymbol{\beta}\in[-\boldsymbol{C},\boldsymbol{C}]} \quad & -\frac{1}{2}\sum_{m,n=1}^{N}\beta_m\beta_n K(\mathbf{x}_m,\mathbf{x}_n) - \epsilon\sum_{n=1}^{N}|\beta_n| + \sum_{n=1}^{N}y_n\beta_n \\ \text{subject to} \quad & \sum_{n=1}^{N}\beta_n = 0, \quad \beta_n \in [-C,C], \quad n = 1\cdots,N. \end{aligned}$$

*Then $h(\mathbf{x}) = \sum_{n=1}^{N}\beta_n^* K(\mathbf{x}_n,\mathbf{x}) + b^*$, where $b^*$ is chosen such that $h(\mathbf{x}_n) - y_n = -\epsilon$ for any $n$ with $0 < \beta_n^* < C$, is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(\mathbf{x},\mathbf{z})$, that solves the optimisation problem (2.15-2.18) [13].*

The solution to the optimisation problem in Proposition 15 therefore minimises the bound on the risk in Proposition 14 for linear $\epsilon$-insensitive loss function

### 2.3.6 Support Vector Regression with Quadratic $\epsilon$-Insensitive Loss

Theorem 14 suggests to optimise the generalisation of the regressor by minimising

$$R^2\|\mathbf{w}\|_2^2 + \sum_{n=1}^{N}L_2^{(\epsilon)}(y_n,\langle\mathbf{w},\mathbf{x}_n\rangle + b),$$

The support vector approach is to minimise

$$\frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C}{2}\sum_{n=1}^{N}L_2^{(\epsilon)}(y_n,\langle\mathbf{w},\mathbf{x}_n\rangle + b)$$

and vary $C$ as discussed above to arrive at the optimal solution. This can be formulated as a quadratic programme

$$\min_{\mathbf{w}\in\mathbb{R}^{n_x},b\in\mathbb{R}} \quad \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C}{2}\sum_{n=1}^{N}(\xi_n^2 + \check{\xi}_n^2), \tag{2.26}$$

$$\text{subject to} \quad (\langle \mathbf{w}, \mathbf{x}_n\rangle + b) - y_n \leq \epsilon + \check{\xi}_n, \tag{2.27}$$

$$y_n - (\langle \mathbf{w}, \mathbf{x}_n\rangle + b) \leq \epsilon + \xi_n, \tag{2.28}$$

$$\xi_n, \check{\xi}_n \geq 0, \qquad n = 1, \cdots, N, \tag{2.29}$$

where the two slack variables $\xi_n, \check{\xi}_n$ are for the cases of exceeding the target value by $\epsilon$, and for being more than $\epsilon$ below the target value, respectively. Thus, they must satisfy

$$\xi_n\check{\xi}_n = 0, \quad \alpha_n\check{\alpha}_n = 0,$$

as $\xi_n\check{\xi}_n \neq 0$ or $\alpha_n\check{\alpha}_n \neq 0$ would require non-zero slack variables in both directions. Notice that if $\xi_n < 0$ or $\check{\xi}_n < 0$, the first two constraints will still hold if we set $\xi_n$,$\check{\xi}_n = 0$, respectively, while this will reduce the value of the objective function. Hence, we can remove the positivity constraints on $\xi_n$,$\check{\xi}_n$ without changing the solution of the optimisation problem. The primal Lagrangian of problem (2.26-2.29) is

$$L_P(\mathbf{w}, b, \boldsymbol{\xi}, \check{\boldsymbol{\xi}}, \boldsymbol{\alpha}, \check{\boldsymbol{\alpha}}) = \frac{1}{2}\|\mathbf{w}\|^2 + \frac{C}{2}\sum_{n=1}^{N}(\xi_n^2 + \check{\xi}_n^2) - \sum_{n=1}^{N}\check{\alpha}_n(\epsilon + \check{\xi}_n + y_n - \langle \mathbf{w}, \mathbf{x}_n\rangle - b) + \tag{2.30}$$

$$- \sum_{n=1}^{N}\alpha_n(\epsilon + \xi_n - y_n + \langle \mathbf{w}, \mathbf{x}_n\rangle + b)). \tag{2.31}$$

Since the solution of this problem has to be a saddle point (Section 2.4), it follows that the partial derivatives of $L_P(\mathbf{w}, b, \boldsymbol{\xi}, \check{\boldsymbol{\xi}}, \boldsymbol{\alpha}, \check{\boldsymbol{\alpha}})$ with respect to the primal variables $\mathbf{w}$, $b$, $\xi_n$, $\check{\xi}_n$ have to vanish,

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^{N}(\alpha_n - \check{\alpha}_n)\mathbf{x}_n = \mathbf{0} \qquad \Rightarrow \mathbf{w} = \sum_{n=1}^{N}(\alpha_n - \check{\alpha}_n)\mathbf{x}_n \tag{2.32}$$

$$\frac{\partial L_P}{\partial \xi_n} = C\xi_n - \alpha_n = 0 \qquad \Rightarrow \xi_n = \frac{\alpha_n}{C} \tag{2.33}$$

$$\frac{\partial L_P}{\partial \check{\xi}_n} = C\check{\xi}_n - \check{\alpha}_n = 0 \qquad \Rightarrow \check{\xi}_n = \frac{\check{\alpha}_n}{C} \tag{2.34}$$

$$\frac{\partial L_P}{\partial b} = \sum_{n=1}^{N}(\check{\alpha}_n - \alpha_n) = 0. \tag{2.35}$$

By substituting back into (2.30) we obtain the dual objective function

$$
\begin{aligned}
& W(\boldsymbol{\alpha}, \check{\boldsymbol{\alpha}}) = \\
=& \frac{1}{2} \sum_{m,n=1}^{N} (\alpha_m - \check{\alpha}_m)(\alpha_n - \check{\alpha}_n)\langle \mathbf{x}_m, \mathbf{x}_n \rangle + \frac{C}{2} \sum_{n=1}^{N} \left( \left( \frac{\alpha_n}{C} \right)^2 + \left( \frac{\check{\alpha}_n}{C} \right)^2 \right) - \epsilon \sum_{n=1}^{N} (\alpha_n + \check{\alpha}_n) \\
& - \sum_{n=1}^{N} (\alpha_n \frac{\alpha_n}{C} + \check{\alpha}_n \frac{\check{\alpha}_n}{C}) + \sum_{n=1}^{N} y_n (\alpha_n - \check{\alpha}_n) - \sum_{m,n=1}^{N} (\alpha_m - \check{\alpha}_m)(\alpha_n - \check{\alpha}_n)\langle \mathbf{x}_m, \mathbf{x}_n \rangle + \\
& + b \sum_{n=1}^{N} (\check{\alpha}_n - \alpha_n) = \\
=& - \frac{1}{2} \sum_{m,n=1}^{N} (\alpha_m - \check{\alpha}_m)(\alpha_n - \check{\alpha}_n)\langle \mathbf{x}_m, \mathbf{x}_n \rangle - \epsilon \sum_{n=1}^{N} (\check{\alpha}_n + \alpha_n) + \sum_{n=1}^{N} y_n (\alpha_n - \check{\alpha}_n) \\
& - \frac{1}{2C} (\langle \boldsymbol{\alpha}, \boldsymbol{\alpha} \rangle + \langle \check{\boldsymbol{\alpha}}, \check{\boldsymbol{\alpha}} \rangle) = \\
=& - \frac{1}{2} \sum_{m,n=1}^{N} (\alpha_m - \check{\alpha}_m)(\alpha_n - \check{\alpha}_n) \left( \langle \mathbf{x}_m, \mathbf{x}_n \rangle + \frac{1}{C} \delta_{mn} \right) - \epsilon \sum_{n=1}^{N} (\alpha_n + \check{\alpha}_n) + \\
& + \sum_{n=1}^{N} y_n (\alpha_n - \check{\alpha}_n)
\end{aligned}
$$

that has to be maximised with respect to the dual variables $\alpha_n, \check{\alpha}_n$. The dual optimisation problem is therefore

$$
\begin{aligned}
\max_{\boldsymbol{\alpha}, \check{\boldsymbol{\alpha}} \geq \mathbf{0}} \quad & - \frac{1}{2} \sum_{m,n=1}^{N} (\alpha_m - \check{\alpha}_m)(\alpha_n - \check{\alpha}_n) \left( \langle \mathbf{x}_m, \mathbf{x}_n \rangle + \frac{1}{C} \delta_{mn} \right) - \epsilon \sum_{n=1}^{N} (\alpha_n + \check{\alpha}_n) + \\
& + \sum_{n=1}^{N} y_n (\alpha_n - \check{\alpha}_n) \\
\text{subject to} \quad & \sum_{n=1}^{N} (\check{\alpha}_n - \alpha_n) = 0, \quad \alpha_n, \, \check{\alpha}_n \geq 0.
\end{aligned}
$$

This is still a quadratic programme. The corresponding KKT complementary conditions (Theorem 21, Section 2.4) are

$$
\begin{aligned}
\check{\alpha}_n \left( \langle \mathbf{w}, \mathbf{x}_n \rangle + b - y_n - \epsilon - \check{\xi}_n \right) &= 0, \\
\alpha_n \left( y_n - \langle \mathbf{w}, \mathbf{x}_n \rangle + b - \epsilon - \xi_n \right) &= 0, \\
\xi_n \check{\xi}_n = 0, \qquad & \alpha_n \check{\alpha}_n = 0, \quad n = 1, \cdots, N.
\end{aligned}
$$

Therefore, only samples with $|h(\mathbf{x}_n) - y_n| \geq \epsilon$, hence not lying inside of the $\epsilon$-tube, have corresponding non-zero multipliers. As a consequence, the weight vector (2.32) of the solution has again a sparse representation in terms of support vectors, and the final hypothesis is of form (2.25),

$$
h(\mathbf{x}) = \sum_{n \in \mathcal{I}_{\text{sv}}} (\alpha_n^* - \check{\alpha}_n^*)\langle \mathbf{x}_n, \mathbf{x} \rangle + b^*.
$$

By substituting $\boldsymbol{\beta} = \boldsymbol{\alpha} - \check{\boldsymbol{\alpha}}$ for our original multipliers $\alpha_n, \check{\alpha}_n$ we can rewrite the problem in a way more resembling the classification case:

$$\begin{aligned} \max_{\boldsymbol{\beta} \in \mathbb{R}^{n_x}} \quad & -\frac{1}{2} \sum_{m,n=1}^{N} \beta_m \beta_n \left( \langle \mathbf{x}_m, \mathbf{x}_n \rangle + \frac{1}{C} \delta_{mn} \right) - \epsilon \sum_{n=1}^{N} |\beta_n| + \sum_{n=1}^{N} y_n \beta_n \\ \text{subject to} \quad & \sum_{n=1}^{N} \beta_n = 0, \quad n = 1 \cdots, N. \end{aligned}$$

For $y_n \in \{-1, 1\}$ and $\epsilon = 0$, the similarity becomes even more apparent with ${\beta'}_n = y_n \beta_n$. The only difference is that ${\beta'}_n$ is not constrained to be positive any more, and that for non-zero $\epsilon$, an additional weight decay factor involving the dual parameters is introduced. In the following proposition, we summarises the support vector algorithm for regression with quadratic $\epsilon$-insensitive loss. We have moved directly to a more general hypothesis space by replacing all inner products appearing in the algorithm with a kernel function $K(\cdot, \cdot)$ calculating the inner product in some feature space (Section 2.5).

**Proposition 16 (SVR with Quadratic $\epsilon$-Insensitive Loss).** *Suppose that we wish to perform regression on a training sample $S = ((\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N))$ and suppose the parameters $\boldsymbol{\beta}^*$ solve the following optimisation problem:*

$$\begin{aligned} \max_{\boldsymbol{\beta} \in \mathbb{R}^{n_x}} \quad & -\frac{1}{2} \sum_{m,n=1}^{N} \beta_m \beta_n \left( K(\mathbf{x}_m, \mathbf{x}_n) + \frac{1}{C} \delta_{mn} \right) - \epsilon \sum_{n=1}^{N} |\beta_n| + \sum_{n=1}^{N} y_n \beta_n \\ \text{subject to} \quad & \sum_{n=1}^{N} \beta_n = 0, \, n = 1 \cdots, N. \end{aligned}$$

*Then $h(\mathbf{x}) = \sum_{n=1}^{N} \beta_n^* K(\mathbf{x}_n, \mathbf{x}) + b^*$, where $b^*$ is chosen such that $h(\mathbf{x}_n) - y_n = -\epsilon - \frac{\beta_n^*}{C}$ for any $n$ with $\beta_n^* > 0$, is equivalent to the hyperplane in the feature space implicitly defined by the kernel $K(\mathbf{x}, \mathbf{z})$, that solves the optimisation problem (2.26-2.29) [13].*

The solution to the optimisation problem in Proposition 16 therefore minimises the bound on the risk in Proposition 14 for quadratic $\epsilon$-insensitive loss function.

The case $\epsilon = 0$ corresponds to considering standard least squares regression with a weight decay factor controlled by $C$, also known as ridge regression. As $C \to \infty$, the problem becomes an unconstrained least square.

### 2.3.7 Notes on the Support Vector Regression Machines

The standard support vector regression algorithms for linear and quadratic $\epsilon$-insensitive loss (Proposition 15, Proposition 16) both possess the following key properties.

1. The hypothesis that has lowest risk is found as the solution of a quadratic programme. This quadratic programme can be solved efficiently. The solution is unique.

2. The weight vector of the final hypothesis is a linear combination of only a small fraction of training vectors, termed support vectors. Therefore, the solution has a sparse representation. The number of support vectors depends on the training data $S$, the parameters $C$ and $\epsilon$, the kernel and the kernel parameters.

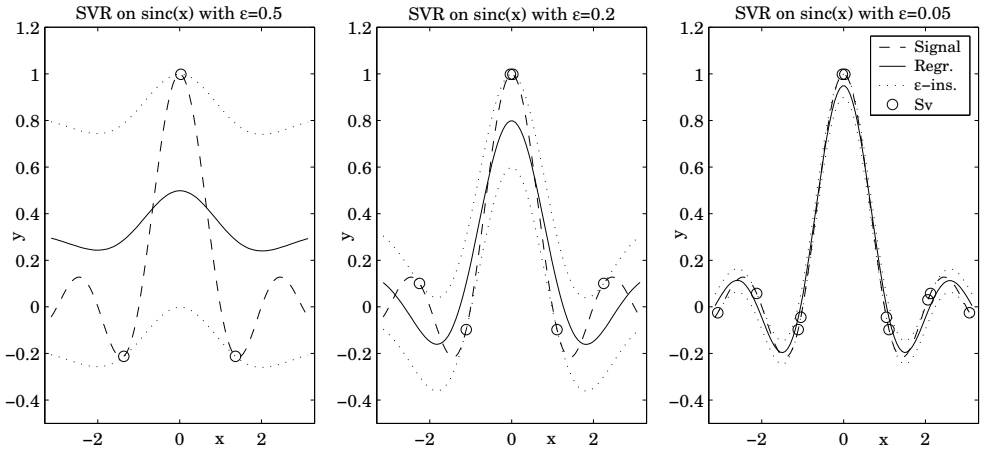


Figure 2.10: *Regression example on* $y = \frac{\sin(x)}{x}$ *without noise, employing a linear* $\epsilon$*-insensitive loss (from left to right:* $\epsilon = 0.5$*,* $\epsilon = 0.2$*,* $\epsilon = 0.05$*) and a Gaussian kernel. The trade off parameter* $C$ *is set to a large value, such that training errors are heavily penalised. The SVM selects the flattest possible hypothesis (dashed - true signal; solid - SVR regressor; dotted -* $\epsilon$*-insensitive region; circles - support vectors).*

3. In both the training algorithm (Proposition 15, 16) and in the final hypothesis of form (2.25), the data enters only inside of inner products $\langle \cdot, \cdot \rangle$. This allows for a generalisation to a rich nonlinear hypothesis space by replacing the inner products by a kernel (Section 2.5).

Whereas the hypothesis selected by the support vector classifiers minimises the risk as a consequence of having large margin on the (reduced) training set, such an interpretation is not possible for the support vector regressors. The solution hypothesis to the SVR algorithm is characterised by being *flat*, i.e. having low capacity, and by forcing the norm of the slack vector to be small, i.e., having low error on the training set (Figure 2.10). Such a hypothesis minimises the bounds on the risk given in Theorem 14.

We note that the only difference between the support vector algorithms with linear and quadratic $\epsilon$-insensitive loss function is that

1. with a linear loss, the dual variables $\beta_n$ are subject to an additional constraint $|\beta_n| \leq C$ (*box constraint*), and

2. with a quadratic loss, $\frac{1}{C}$ is added to the diagonal of the kernel matrix $\boldsymbol{K}$ with entries $K_{mn} = K(\mathbf{x}_m, \mathbf{x}_n)$.

Figure 2.11 shows a typical support vector regression on a $\frac{\sin(x)}{x}$ - dependency hidden in additive white Gaussian noise, produced with a Gaussian kernel. The support vectors (circles) are the 37 training samples that happen to lie outside or on the border of the $\epsilon$-insensitive region around the hypothesis.

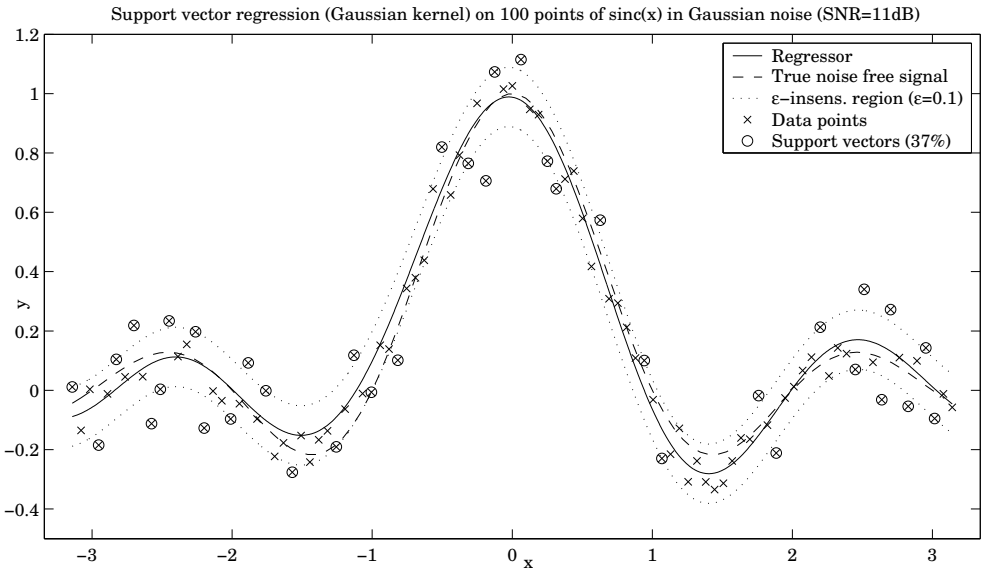


Figure 2.11: *Regression example on* $y = \frac{\sin(x)}{x}$ *in additive white noise, employing a linear* $\epsilon$*-insensitive loss and a Gaussian kernel.*

### 2.3.8 Solving the Optimisation Problems

Training the SVMs requires solving the optimisation problems in Propositions 10, 11, 12, 15 and 16. The solution involves two steps.

1. First, the entries $G_{mn} = \langle \mathbf{x}_m, \mathbf{x}_n \rangle$ of the Gram matrix or, when working in feature space, the entries $K_{mn} = K(\mathbf{x}_m, \mathbf{x}_n)$ of the kernel matrix have to be computed. Given $N$ training vectors, this requires $\frac{N(N+1)}{2}$ computations of the inner product between two vectors or evaluations of the kernel. One single inner product calculation requires $\mathcal{O}(n_x)$ operations ($n_x$ multiplications and $n_x - 1$ additions), where $n_x$ is the dimension of the input space $X$. The complexity of the kernel evaluation depends on the type of kernel that is used. The evaluation of inner product type kernels such as the polynomial kernel and of RBF kernels such as the Gaussian kernel requires $\mathcal{O}(n_x)$ operations. Therefore, the overall complexity when employing such kernels or when working in input space is $\mathcal{O}(N^2, n_x)$.

2. Then, the specific quadratic optimisation problems are fed into a quadratic programming problem solver. The support vector optimisation problem can be solved analytically. The analytical solution requires in the worst case $\mathcal{O}(N_{\text{sv}}^3)$ operations, where $N_{\text{sv}}$ is the number of support vectors, and is therefore only feasible for a small number of training samples [5]. There exists a variety of strategies for solving quadratic optimisation problems with severely less computational and memory requirements than the analytical solution, such as interior point algorithms and active set algorithms. They exploit specific properties of the support vector optimisation problems and are feasible for $N$ and $N_{\text{sv}}$ being larger than 100000.

We do not aim at exploring the relative merits of different methods for solving the quadratic programming problems here and refer to [5], [13] and [61] for overviews, and

to [20] for a detailed exposition on optimisation techniques.

### 2.3.9 Extensions: Linear Programming and $\nu$-SVR Machine

A number of variations and extensions to the standard SVMs described in the previous sections have been considered. We will shortly review two of them, namely the linear programming SVM and the $\nu$-SVM.

**The Linear Programming SVM.** Linear programming SVMs select a hypothesis $h(\mathbf{x}) = \sum_{n=1}^{N} \beta_n \langle \mathbf{x}_n, \mathbf{x} \rangle + b^*$ that minimises a regularised risk functional of form

$$R_{\text{reg}} = R_{\text{emp}} + \lambda \|\boldsymbol{\beta}\|_1^2.$$

Generalisation bounds motivating a risk functional of this type can be derived (see e.g. [62] and [76]). The difference to standard SVMs is that the algorithms of linear programming SVMs reduce to a linear optimisation problem, which is easier to solve than the standard support vector algorithms.

**The $\nu$-SVM.** Schölkopf et al. proposed to optimise the width of the $\epsilon$-insensitive zone within the training algorithm [53] [54]. A constant $\nu \geq 0$ is introduced and defines a trade off between $\epsilon$, the model complexity and the slack variables. The resulting modified optimisation problem (with linear $\epsilon$-insensitive loss) is

$$\begin{aligned} \min_{\mathbf{w} \in \mathbb{R}^{n_x}, b \in \mathbb{R}, \epsilon \in \mathbb{R}^+} \quad & \|\mathbf{w}\|_2^2 + C \left( \nu \epsilon + \frac{1}{N} \sum_{n=1}^{N} (\xi_n + \check{\xi}_n) \right), \\ \text{subject to} \quad & (\langle \mathbf{w}, \mathbf{x}_n \rangle + b) - y_n \leq \epsilon + \check{\xi}_n, \quad n = 1, \cdots, N, \\ & y_n - (\langle \mathbf{w}, \mathbf{x}_n \rangle + b) \leq \epsilon + \xi_n, \quad n = 1, \cdots, N, \\ & \xi_n, \check{\xi}_n, \epsilon \geq 0, \quad n = 1, \cdots, N, \end{aligned}$$

The solution to this problem is the saddle point of the primal Lagrangian

$$L_P(\mathbf{w}, b, \boldsymbol{\xi}, \check{\boldsymbol{\xi}}, \boldsymbol{\alpha}, \check{\boldsymbol{\alpha}}, \beta, \epsilon) = \frac{1}{2} \|\mathbf{w}\|_2^2 + \frac{C}{N} \sum_{n=1}^{N} (\xi_n + \check{\xi}_n) - \sum_{n=1}^{N} \alpha_n (\epsilon + \xi_n - y_n + \langle \mathbf{w}, \mathbf{x}_n \rangle + b) - \tag{2.36}$$

$$- \beta \epsilon - \sum_{n=1}^{N} \check{\alpha}_n (\epsilon + \check{\xi}_n + y_n - \langle \mathbf{w}, \mathbf{x}_n \rangle - b) - \sum_{n=1}^{N} (r_n \xi_n + \check{r}_n \check{\xi}_n) + C \nu \epsilon \tag{2.37}$$

and is found by forcing the derivatives with respect to the primal variables $\mathbf{w}, b, \boldsymbol{\xi}, \check{\boldsymbol{\xi}}, \epsilon$ to be zero, yielding the equations

$$\begin{aligned}
\frac{\partial L_P}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{n=1}^{N}(\alpha_n - \check{\alpha}_n)\mathbf{x}_n = \mathbf{0} \\
\frac{\partial L_P}{\partial \epsilon} &= C\nu - \sum_{n=1}^{N}(\alpha_n + \check{\alpha}_n) - \beta = 0 \\
\frac{\partial L_P}{\partial \xi_n} &= \frac{C}{N} - \alpha_n - r_n = \frac{\partial L_P}{\partial \check{\xi}_n} = \frac{C}{N} - \check{\alpha}_n - \check{r}_n = 0 \\
\frac{\partial L_P}{\partial b} &= \sum_{n=1}^{N}(\check{\alpha}_n - \alpha_n) = 0,
\end{aligned}$$

and substituting into the primal (2.36). The $\nu$-SVR algorithm is then given by the optimisation problem

$$\max_{\boldsymbol{\alpha}, \check{\boldsymbol{\alpha}} \in [\mathbf{0}, \frac{C}{N}]} \quad -\frac{1}{2}\sum_{m,n=1}^{N}(\alpha_m - \check{\alpha}_m)(\alpha_n - \check{\alpha}_n)\langle \mathbf{x}_m, \mathbf{x}_n \rangle + \sum_{n=1}^{N} y_n(\alpha_n - \check{\alpha}_n) \tag{2.38}$$

$$\text{subject to} \quad \sum_{n=1}^{N}(\check{\alpha}_n - \alpha_n) = 0, \quad \sum_{n=1}^{N}(\alpha_n + \check{\alpha}_n) \leq C\nu, \quad \alpha_n,\, \check{\alpha}_n \in [0, \frac{C}{N}]. \tag{2.39}$$

This is again a quadratic programme and has a sparse solution of form (2.25) in terms of support vectors. Schölkopf et al. proved the following properties of the $\nu$-SVR algorithm [54].

1. $\nu$ is an upper bound on the fraction of errors on the training set.

2. $\nu$ is a lower bound on the fraction of support vectors.

3. If the data are generated i.i.d. from a distribution $P(\mathbf{x}, y) = P(\mathbf{x})P(y|\mathbf{x})$ with $P(y|\mathbf{x})$ continuous, $\nu$ asymptotically equals both the fraction of errors on the training set and the number or support vectors.

It can be show that the optimal value for $\epsilon$ scales linearly with the variance of the noise [33]. Thus, the $\nu$-SVR algorithm can be interpreted as automatically adapting to the noise level. Still, one has to consider the optimal choice of the parameter $\nu$ [9].

### 2.3.10 Maximum Likelihood Estimators and Loss Functions

Till now, the SVR algorithms seem to be rather strange and hardly related to other existing methods for function estimation. We show now the connection between loss functions and maximum likelihood estimators, as discussed in many statistics textbooks, mainly following [60] and [61].

**Maximum Likelihood Estimators.** Assume the data $S = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}$ to be generated i.i.d. by an underlying functional dependency $t(\mathbf{x})$, and additive noise with density $p(\xi)$ that is independent of $\mathbf{x}$,

$$\begin{aligned} y_n &= t(\mathbf{x}_n) + \xi_n, \\ P(y|\mathbf{x}) &= p(y - t(\mathbf{x}))P(\mathbf{x}). \end{aligned}$$

Then, the likelihood $P(S|h(\mathbf{X}))$ of $S$ given $h(\mathbf{X}) = \{h(\mathbf{x}_1), \cdots, h(\mathbf{x}_N)\}$ is

$$P(S|h(\mathbf{X})) = \prod_{n=1}^{N} p(y_n - h(\mathbf{x}_n)). \tag{2.40}$$

Rewriting

$$p(y - h(\mathbf{x})) = e^{-L(y-h(\mathbf{x}))},$$

for some function $L(\cdot)$, the likelihood $P(S|h(\mathbf{X}))$ becomes

$$P(S|h(\mathbf{X})) = e^{\sum_{n=1}^{N} L(y_n - h(\mathbf{x}_n))}.$$

Maximising (2.40) is equivalent to minimising

$$-\log P(S|h(\mathbf{X})) = \sum_{n=1}^{N} L(y_n - h(\mathbf{x}_n)).$$

Hence, the optimal loss function in the maximum likelihood sense is the function

$$L(y, h(\mathbf{x})) = -\log p(y - h(\mathbf{x})). \tag{2.41}$$

**Common Loss Functions.** The optimal loss function in the maximum likelihood sense is given by (2.41). This allows to choose an appropriate loss function if one has knowledge of the noise distribution in a problem. The loss functions arising from a given noise distribution through (2.41) may, however, be non-convex. In the derivation of the support vector algorithms, only the quadratic loss, the Laplacian loss, the Huber loss and the linear and quadratic $\epsilon$-insensitive losses lead to quadratic programming problems, whereas other loss functions lead to optimisation problems that are more difficult to solve [60]. For a specific loss function from some real world problem, we can still try to find a convex approximation for the loss function, leading to a convex optimisation problem. Note, however, that only loss functions with an $\epsilon$-insensitive region will produce a sparse representation of form (2.25) for the hypothesis. Table 2.1 summarises the loss functions for which the support vector algorithms are quadratic programming problems with corresponding density models. They are illustrated in Figure 2.12 for $C = \epsilon = \sigma = 1$.

Huber [28] showed that if one only knows that the density describing the noise is a convex function possessing second derivatives, the best approximation for the worst possible density is the Laplacian loss function $L(y, h(\mathbf{x})) = |y - h(\mathbf{x})|$. The linear $\epsilon$-insensitive loss function can be seen as a generalisation of this loss function by introducing an insensitive zone. Thus, if one knows few about the noise density, the linear $\epsilon$-insensitive loss is a good choice for robust regression. In all experiments reported in this work, we employ SVMs with linear $\epsilon$-insensitive loss functions.

| | Loss function | Density model |
|---|---|---|
| Linear $\epsilon$-ins. | $L_1^{(\epsilon)}(\eta) = \begin{cases} 0 & \lvert\eta\rvert < \epsilon \\ C\left(\lvert\eta\rvert - \epsilon\right) & \lvert\eta\rvert \geq \epsilon \end{cases}$ | $p(\eta) = \begin{cases} \frac{C}{2(1+C\epsilon)} & \lvert\eta\rvert < \epsilon \\ \frac{C}{2(1+C\epsilon)} \mathrm{e}^{(-C(\lvert\eta\rvert - \epsilon))} & \lvert\eta\rvert \geq \epsilon \end{cases}$ |
| Laplacian | $C\lvert\eta\rvert$ | $p(\eta) = \frac{C}{2}\mathrm{e}^{(-C\lvert\eta\rvert)}$ |
| Quadratic $\epsilon$-ins. | $L_2^{(\epsilon)}(\eta) = \begin{cases} 0 & \lvert\eta\rvert < \epsilon \\ \frac{C}{2}\left(\lvert\eta\rvert - \epsilon\right)^2 & \lvert\eta\rvert \geq \epsilon \end{cases}$ | $p(\eta) = \begin{cases} \frac{\sqrt{C}}{2\epsilon\sqrt{C}+\sqrt{2\pi}} & \lvert\eta\rvert < \epsilon \\ \frac{\sqrt{C}}{2\epsilon\sqrt{C}+\sqrt{2\pi}} \mathrm{e}^{(-\frac{C}{2}(\lvert\eta\rvert - \epsilon)^2)} & \lvert\eta\rvert \geq \epsilon \end{cases}$ |
| Gaussian | $\frac{C}{2}\eta^2$ | $p(\eta) = \frac{\sqrt{C}}{\sqrt{2\pi}} \mathrm{e}^{\left(-\frac{C\eta^2}{2}\right)}$ |
| Huber's loss | $\begin{cases} \frac{C}{2\sigma}\eta^2 & \lvert\eta\rvert \leq \sigma \\ C\left(\lvert\eta\rvert - \frac{\sigma}{2}\right) & \text{otherwise} \end{cases}$ | $p(\eta) \propto \begin{cases} \mathrm{e}^{\left(-\frac{C\eta^2}{2\sigma}\right)} & \lvert\eta\rvert \leq \sigma \\ \mathrm{e}^{(C\left(\frac{\sigma}{2} - \lvert\eta\rvert\right))} & \text{otherwise} \end{cases}$ |

Table 2.1: *Loss functions and corresponding density models. The trade off parameter $C > 0$ that appears in the SVR algorithms is treated as an additional linear scaling in the loss functions (Footnote 4).*

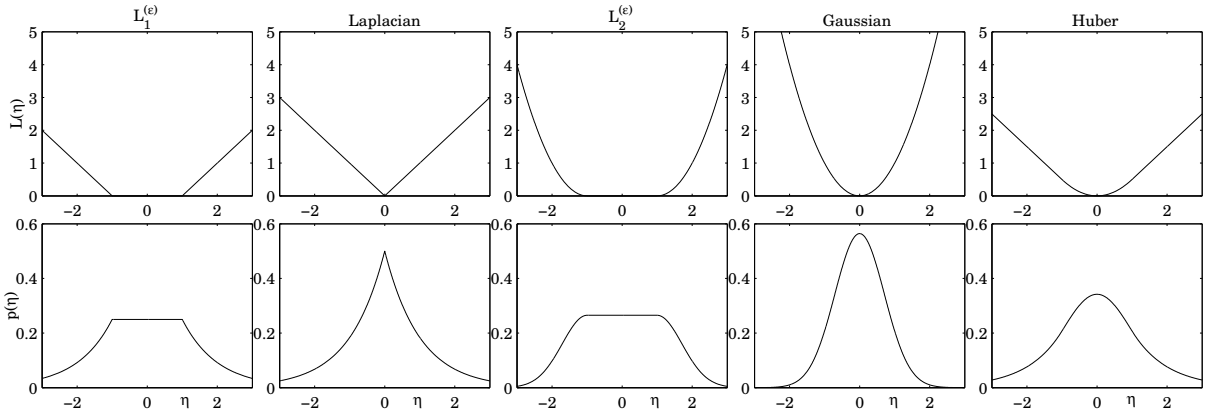


Figure 2.12: *Common loss functions $L(\eta)$ (top) and corresponding density models $p(\eta)$ (bottom) from Table 2.1, with parameters $C = \epsilon = \sigma = 1$.*

## 2.4 Optimisation Theory

In this section, we summarise the results from optimisation theory that we use for the development of the support vector algorithms, mainly following [13] and [20]. Starting with general definitions and notations for optimisation problems, we specialise them for our needs, and move on towards Lagragian theory, which characterises the solution of such problems. The final result, the Theorem of Kuhn-Tucker, provides conditions on the solution of the support vector optimisation problems.

### 2.4.1 The Optimisation Problem

The support vector solution can be found as the minimum of some function, subject to constraints. This can be formulated as an optimisation problem. The general optimisation

problem is of form

$$\min_{\mathbf{w}\in\Omega} \quad f(\mathbf{w}) \tag{2.42}$$

$$\text{subject to} \quad g_i(\mathbf{w}) \leq 0, \quad i = 1, \cdots, k, \tag{2.43}$$

$$h_i(\mathbf{w}) = 0, \quad i = 1, \cdots, m, \tag{2.44}$$

with $f(\mathbf{w})$ the *objective function*, and $g_i(\mathbf{w})$ and $h_i(\mathbf{w})$ the *inequality* and *equality constraints*, respectively. The functions $f(\mathbf{w})$, $\mathbf{g}(\mathbf{w})$ and $\mathbf{h}(\mathbf{w})$ are defined on a domain $\Omega \subseteq \mathbb{R}^n$, and the subset $\Omega_F \subseteq \Omega$ on which both the function $f(\mathbf{w})$ is defined, and the constraints are satisfied, is called the *feasible region*. An inequality constraint $g_i(\mathbf{w}) \leq 0$ is said to be *active* if the solution $\mathbf{w}^*$ satisfies $g_i(\mathbf{w}^*) = 0$, otherwise it is said to be *inactive*.

A point $\mathbf{w}^* \in \Omega_F$ such that there exists no other point $\mathbf{w} \in \Omega_F$ for which $f(\mathbf{w}) < f(\mathbf{w}^*)$, is a *solution*, or global minimum, of the problem (2.42-2.44). A *local minimum* is a point $\mathbf{w}^* \in \Omega_F$ such that there exists $\epsilon > 0$, $\forall \mathbf{w} \in \Omega_F$, $f(\mathbf{w}) \geq f(\mathbf{w}^*)$: $\|\mathbf{w} - \mathbf{w}^*\| < \epsilon$.

We can make further statements on the solution of the general problem (2.42-2.44) if the real-valued objective function $f(\mathbf{w})$ is convex for $\mathbf{w} \in \mathbb{R}^n$, and therefore, for any $\theta \in (0, 1)$, $\forall \mathbf{w}, \mathbf{u} \in \mathbb{R}^n$, satisfies

$$f(\theta\mathbf{w} + (1-\theta)\mathbf{u}) \leq \theta f(\mathbf{w}) + (1-\theta) f(\mathbf{u}).$$

Then, any local minimum $\mathbf{w}^*$ of $f$ is also a global minimum [13].

The problem (2.42-2.44) is called a *convex optimisation problem* if the set $\Omega$, the objective function and all constraints are convex. In the special case where the objective function $f$ is quadratic[13] and all constraints $\boldsymbol{g}$, $\boldsymbol{h}$ are linear, the problem (2.42-2.44) is a *quadratic programme*. We will see that for convex optimisation problems, solving the primal and the dual Lagrangian problem is equivalent and leads to the same solution. The solution is unique and is a saddle point.

**The Support Vector Optimisation Problems.** All problems arising in the derivation of the support vector algorithms are problems defined over the convex domain $\Omega = \mathbb{R}^n$, with linear constraints that can be expressed in terms of affine functions $\boldsymbol{y}(\mathbf{w}) = \mathbf{A}\mathbf{w} + \mathbf{b}$, and a convex and quadratic objective function [13]. Therefore, we are confronted with quadratic programmes.

### 2.4.2 Solving the Optimisation Problem

The solution of an optimisation problem (2.42-2.44) is characterised by Lagrangian theory. We restrict ourselves here to the special case of quadratic programmes.

**Unconstrained Problem.** The solution of the unconstrained problem (2.42) is characterised by the stationarity of the objective function.

**Theorem 17 (Fermat).** *A necessary condition for* $\mathbf{w}^*$ *to be a minimum of* $f(\mathbf{w})$, $f \in C^1$, *is* $\frac{\partial f(\mathbf{w}^*)}{\partial \mathbf{w}} = \mathbf{0}$. *If* $f$ *is convex, this condition is also sufficient (e.g. in [13]).*

[13] A quadratic function is a function of form $f(\mathbf{w}) = a\mathbf{w}^T\mathbf{w} + \mathbf{b}^T\mathbf{w} + c, \quad a \neq 0.$

**Equality Constraints.** When there are constraints, we need to find a function that holds information on both the objective function and the constraints, and whose stationarity can serve to detect solutions of the problem. This function is called the *Lagrangian.*

For optimisation problems with only equality constraints,

$$\begin{aligned} \min_{\mathbf{w}\in\Omega\subseteq\mathbb{R}^n} \quad & f(\mathbf{w}) \\ \text{subject to} \quad & h_i(\mathbf{w}) = 0, \quad i = 1, \cdots, m, \end{aligned}$$

the Lagrangian function is defined as

$$L_P(\mathbf{w}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w}),$$

where the coefficients $\beta_i$ are called the Lagrange multipliers.

For $\mathbf{w}^*$ being a local minimum of such a problem, $\frac{\partial f(\mathbf{w}^*)}{\partial \mathbf{w}}$ is possibly non-zero, since moving in the directions in which we could reduce the objective function may cause us to violate one or more of the constraints. For fulfilling all of them, we can only move perpendicular to $\frac{\partial h_i(\mathbf{w}^*)}{\partial \mathbf{w}}$, and therefore perpendicular to the subspace spanned by $\{\frac{\partial h_i(\mathbf{w}^*)}{\partial \mathbf{w}} : i = 1, \cdots, m\}$. For linearly independent $\frac{\partial h_i(\mathbf{w}^*)}{\partial \mathbf{w}}$, no move within the feasible region can reduce the objective function if $\frac{\partial f(\mathbf{w}^*)}{\partial \mathbf{w}}$ lies in this subspace, $\frac{\partial f(\mathbf{w}^*)}{\partial \mathbf{w}} + \sum_{i=1}^{m} \beta_i \frac{\partial h_i(\mathbf{w}^*)}{\partial \mathbf{w}} = 0$. This leads us to the following result, describing the solution of optimisation problems with only equality constraints.

**Theorem 18 (Lagrange).** *A necessary condition for a normal point* $\mathbf{w}^*$ *to be a minimum of* $f(\mathbf{w})$ *subject to* $h_i(\mathbf{w}) = 0$, $i = 1, \cdots, m$, *with* $f, h_i \in C^1$, *is*

$$\begin{aligned} \frac{\partial L_P(\mathbf{w}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{w}} &= \mathbf{0} \\ \frac{\partial L_P(\mathbf{w}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} &= \mathbf{0} \end{aligned}$$

*for some values* $\boldsymbol{\beta}^*$. *The above conditions are sufficient provided that* $L_P(\mathbf{w}, \boldsymbol{\beta}^*)$ *is a convex function of* $\mathbf{w}$ *(e.g. in [13]).*

The solution is obtained by jointly solving the two systems, where the first one gives a new system of equations, and the second one returns the equality constraints.

**Equality and Inequality Constraints.** For an optimisation problems that is subject to both equality and inequality constraints,

$$\begin{aligned} \min_{\mathbf{w}\in\Omega\subseteq\mathbb{R}^n} \quad & f(\mathbf{w}) && (2.45) \\ \text{subject to} \quad & g_i(\mathbf{w}) \leq 0, \quad i = 1, \cdots, k, && (2.46) \\ & h_i(\mathbf{w}) = 0, \quad i = 1, \cdots, m, && (2.47) \end{aligned}$$

the generalised Lagrangian function is defined as

$$L_P(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}) + \sum_{i=1}^{m} \beta_i h_i(\mathbf{w}) = \\ = f(\mathbf{w}) + \boldsymbol{\alpha}^T \mathbf{g}(\mathbf{w}) + \boldsymbol{\beta}^T \mathbf{h}(\mathbf{w}),$$

with Lagrangian multipliers $\alpha_i$ and $\beta_i$. For this problem, we can define a dual problem,

$$\max_{\boldsymbol{\alpha} \in \mathbb{R}^n_+, \boldsymbol{\beta} \in \mathbb{R}^n} \quad \theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{2.48}$$

$$\theta(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \inf_{\mathbf{w} \in \Omega} L_P(\mathbf{w}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{2.49}$$

$$\text{subject to} \quad \boldsymbol{\alpha} \geq \mathbf{0}, \tag{2.50}$$

in which the dual variables $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are considered as the fundamental unknowns. The primal and dual problems (2.45-2.47) and (2.48-2.50) possess specific properties with respect to each other. First, they are connected through the *weak duality theorem*:

**Theorem 19 (Weak Duality Theorem).** *Let $\mathbf{w} \in \Omega$ be a feasible solution of the primal problem (2.45-2.47) and $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ a feasible solution of the dual problem (2.48-2.50). Then $f(\mathbf{w}) \geq \theta(\boldsymbol{\alpha}, \boldsymbol{\beta})$ (e.g. in [13]).*

The difference of the values of the solution of the primal and dual problem is known as the *duality gap*. If the duality gap is zero, $f(\mathbf{w}^*) = \theta(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$, $\mathbf{w}^*$ and $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ solve the primal and the dual problems respectively, and $\alpha_i^* g_i(\mathbf{w}^*) = 0$, for $i = 1, \cdots, k$, since the inequality in the weak duality theorem then becomes an equality. It is not always guaranteed that the solutions of the primal and dual problem have the same value.

For the specific case of an optimisation problem on a convex domain, however, with the constraints being affine functions, the strong duality theorem holds, and the duality gap is zero.

**Theorem 20 (Strong Duality Theorem).** *Given an optimisation problem with convex domain $\Omega \subseteq \mathbb{R}^n$,*

$$\begin{aligned} \min_{\mathbf{w} \in \Omega} \quad & f(\mathbf{w}) \\ \text{subject to} \quad & g_i(\mathbf{w}) \leq 0, \quad i = 1, \cdots, k, \\ & h_i(\mathbf{w}) = 0, \quad i = 1, \cdots, m, \end{aligned}$$

*where $g_i$ and $h_i$ are affine functions, that is $\mathbf{h}(\mathbf{w}) = \mathbf{A}\mathbf{w} - \mathbf{b}$ for some matrix $\mathbf{A}$ and vector $\mathbf{b}$, the duality gap is zero (e.g. in [13]).*

In this case, the solution $(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$, with $\mathbf{w}^* \in \Omega$, $\boldsymbol{\alpha}^* \geq \mathbf{0}$, is a *saddle point* of the primal problem, satisfying

$$L_P(\mathbf{w}^*, \boldsymbol{\alpha}, \boldsymbol{\beta}) \leq L_P(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) \leq L_P(\mathbf{w}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$$

for all $\mathbf{w} \in \Omega$, $\boldsymbol{\alpha} \geq \mathbf{0}$ (*saddle point condition*). Therefore, we can equivalently solve the dual problem instead of the primal problem, and the dual objective function can be

obtained by forcing the partial derivatives of the primal objective function with respect to the primal variables $\mathbf{w}$ to be zero.

We are now in position to quote the final result, the Kuhn-Tucker theorem, providing conditions for the solution to the optimisation problem we are confronted with when training SVMs. It brings together Theorems 17 to 20.

**Theorem 21 (Kuhn-Tucker).** *(e.g. in [13]) Given an optimisation problem with convex domain $\Omega \subseteq \mathbb{R}^n$,*

$$\begin{aligned} \min_{\mathbf{w}\in\Omega} \quad & f(\mathbf{w}) \\ \text{subject to} \quad & g_i(\mathbf{w}) \leq 0, \quad i = 1, \cdots, k, \\ & h_i(\mathbf{w}) = 0, \quad i = 1, \cdots, m, \end{aligned}$$

*with $f \in C^1$ convex and $g_i$, $h_i$ affine, necessary and sufficient conditions for a normal point $\mathbf{w}^*$ to be an optimum are the existence of $\boldsymbol{\alpha}^*$, $\boldsymbol{\beta}^*$ such that*

$$\begin{aligned} \frac{\partial L_P(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \mathbf{w}} &= \mathbf{0}, \\ \frac{\partial L_P(\mathbf{w}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)}{\partial \boldsymbol{\beta}} &= \mathbf{0}, \\ \alpha_i^* g_i(\mathbf{w}^*) &= 0, \quad i = 1, \cdots, k, \\ g_i(\mathbf{w}^*) &\leq 0, \quad i = 1, \cdots, k, \\ \alpha_i^* &\geq 0, \quad i = 1, \cdots, k. \end{aligned}$$

The third relation, known as *Karush-Kuhn-Tucker (KKT) complementary condition*, implies that for active inequality constraints $\alpha_i^* \geq 0$, whereas for inactive constraints the Lagrangian multiplier must be zero. In the first case, with the solution point being on the boundary of the feasible region, the conditions for the optimal solution are given by Lagrange's theorem with non-zero $\alpha_i$, whereas in the second case, with the solution point lying on the inside of the feasible region, Fermat's theorem applies. The KKT complementary conditions imply that for certain optimisations the number of variables involved may be significantly smaller then the full training set size.

Lagrangian treatment of convex optimisation problems leads to an alternative dual description. The primal objective function can be transformed into the dual objective function by setting the derivatives with respect to the primal variables to zero, and substituting back. The resulting objective function depends only on dual variables and must be maximised under simpler constraints.

## 2.5 Kernels and Feature Space

In this section, we clarify how the SVM algorithms can be made nonlinear by replacing the inner products with a kernel function. The exposition will mainly follow [13]. Since introductory literature on the theory of reproducing kernels is rare, we decide to hold the section as self-contained as possible.

### 2.5.1 Explicit Mapping into Feature Space

The representation of a functionality that one wishes to learn largely determines the difficulty of the learning task. For instance, take the training examples $(\mathbf{x}, y)$, $\mathbf{x} \in \mathbb{R}^2$, $y \in \mathbb{R}$ that stem from a functionality $y = ax_1^2 + bx_1x_2 + c$ of monomials of degree 2 of the coordinates of the attributes $\mathbf{x}$, which cannot be learned by a linear machine. This functionality becomes linear under the mapping $\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x}) = (x_1^2, x_1x_2, x_2^2)$.

The strategy of pre-processing the original attributes $\mathbf{x}$, by mapping the input space $X$ into a new space $\mathcal{F} = \{\boldsymbol{\phi}(\mathbf{x}) | \mathbf{x} \in X\}$, called *feature space*,

$$\mathbf{x} = [x_1, \cdots, x_n] \rightarrow \boldsymbol{\phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), \cdots, \phi_v(\mathbf{x})],$$

is common in machine learning since a long time (Figure 2.13 gives an illustration of this strategy). As the number of features $\phi_i$ increases, however, both the computational and

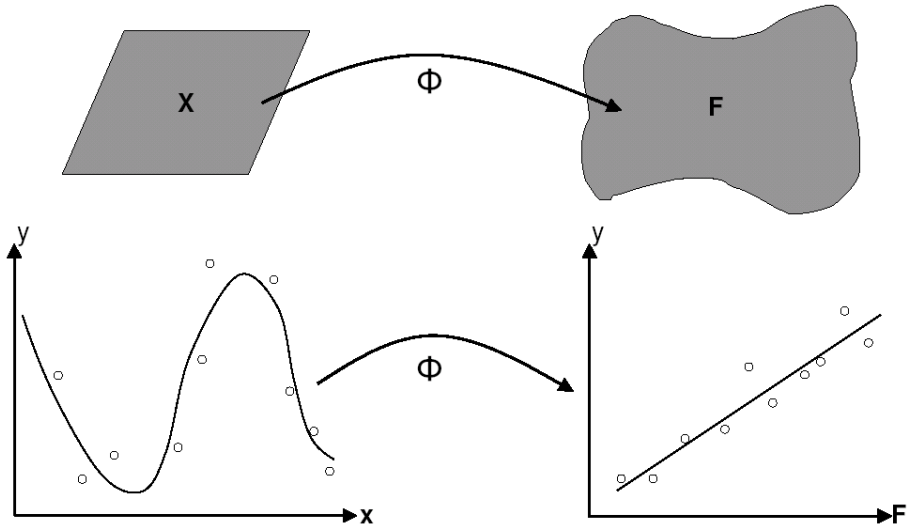


Figure 2.13: *Mapping the input space in a feature space, and working in this feature space, can highly facilitate the learning task. A functionality that can not be learned by a linear machine in input space (left) can become linear in feature space (right).*

the generalisation performance of the learning machine can decrease. The computational cost grows with the number of features, and a large number of features makes the learning machine more flexible such that it can become subject to overfitting. This is problematic, since with a larger set of possibly redundant features it is as well more likely that the target functionality can be learned by a standard machine, a phenomenon often referred to as the *curse of dimensionality*. If, for instance, we consider monomials and features of degree $d$ in our simple example from above, the dimension of the feature space becomes $v = \binom{n_x+d-1}{d}$, which is computationally intractable already for quite small degree $d$ and dimension of the input space $n_x$.

The specific construction of SVMs makes it possible to go around the curse of dimensionality. The problem of generalisation is avoided by using a learning machine with an induction principle that is based on results from statistical learning theory (Section 2.1), and the computational problem is avoided by replacing the explicit mapping strategy with

an implicit one, as will be made clear in the following subsections. This allows SVMs to work even in infinite dimensional feature spaces.

### 2.5.2 Implicit Mapping into Feature Space

In order to be able to learn a nonlinear relation with a linear machine, we need to represent the data through a set of non-linear features $\boldsymbol{\phi} : X \to \mathcal{F}$, so that the set of hypothesis will be of type

$$h(\mathbf{x}) = \sum_{i=1}^{v} \theta_i \phi_i(\mathbf{x}) + b.$$

We have seen in Sections 2.2 and 2.3 that the support vector hypotheses can be represented in a dual form

$$h(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n \langle \boldsymbol{\phi}(\mathbf{x}_n), \boldsymbol{\phi}(\mathbf{x}) \rangle + b,$$

which can be evaluated by using only inner products between input and training vectors. The training vectors enter the SVM training algorithms (Proposition 15, 16) as well only inside of inner products. If there would be a way to calculate the inner products in feature space, $\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z}) \rangle$, directly from the input vectors $\mathbf{x}$ and $\mathbf{z}$ via some function, the computational complexity would depend only on how easy the evaluation of this function is, and not on the dimensionality $v$ of the feature space. Such a function is called a *kernel*.

**Definition 22 (Kernel).** *A kernel is a function $K(\cdot, \cdot) : X \times X \mapsto \mathbb{R}$, such that for all $\mathbf{x}, \mathbf{z} \in X$,*

$$K(\mathbf{x}, \mathbf{z}) = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z}) \rangle,$$

*where $\boldsymbol{\phi}$ is a mapping from $X$ to some inner product feature space $\mathcal{F}$.*

Thus, if we find a kernel that can be evaluated efficiently, the evaluation of our hypothesis

$$h(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n K(\mathbf{x}_n, \mathbf{x}) + b$$

takes at most $N$ kernel computations. Interestingly, the underlying feature map $\boldsymbol{\phi}(\cdot)$ does not even have to be known explicitly. It is implicitly defined by the choice of kernel $K(\cdot, \cdot)$, that calculates the inner product in some feature space $\mathcal{F}$.

Kernels can be viewed as a generalisation of the inner product, and the simplest kernel is the one corresponding to the identity map as the feature map $\boldsymbol{\phi}(\cdot)$,

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle.$$

By defining the feature map as a fixed linear transformation $\mathbf{x} \to \mathbf{A}\mathbf{x}$, the kernel function is

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{A}\mathbf{x}, \mathbf{A}\mathbf{z} \rangle = \mathbf{x}^T \mathbf{A}^T \mathbf{A} \mathbf{z} = \mathbf{x}^T \mathbf{B} \mathbf{z}.$$

Other examples are *polynomial kernels*

$$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x}, \mathbf{z} \rangle^d \text{ and } K(\mathbf{x}, \mathbf{z}) = (\langle \mathbf{x}, \mathbf{z} \rangle + c)^d \tag{2.51}$$

with the $\binom{n_x+d-1}{d}$ and $\binom{n_x+d}{d}$ monomials of $x_j$ and $z_j$ up to degree $d$ as features, respectively, and the *Gaussian kernel*

$$K(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2/\sigma^2) = \exp(-\gamma\|\mathbf{x} - \mathbf{z}\|^2), \tag{2.52}$$

with an infinite number of features.

Finding a kernel that is easy to evaluate is the key issue in the approach of implicitly defining a map into some feature space. Without the use of kernel functions, SVMs - as linear learning machines - would have to use explicit feature maps and suffer from the same computational problems as other methods. It is the pleasing fact that kernels can be naturally introduced in both the hypothesis and the training algorithm that, together with their foundation on statistical learning theory, renders SVMs so powerful.

In the next subsection, we will give a mathematical characterisation of kernels.

### 2.5.3 Characterisation of Kernels

For directly defining a kernel function that is the inner product in some feature space, and therefore implicitly determines a feature map, we need to know more about the general properties of such a function $K(\cdot, \cdot)$. It is obvious that it must be symmetric,

$$K(\mathbf{x}, \mathbf{z}) = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z}) \rangle = \langle \boldsymbol{\phi}(\mathbf{z}), \boldsymbol{\phi}(\mathbf{x}) \rangle = K(\mathbf{z}, \mathbf{x}),$$

and that it necessarily has to satisfy the inequality

$$\begin{aligned} &K(\mathbf{x}, \mathbf{z})^2 = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z}) \rangle^2 \leq \\ &\leq \|\boldsymbol{\phi}(\mathbf{x})\|^2 \cdot \|\boldsymbol{\phi}(\mathbf{z})\|^2 = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}) \rangle \langle \boldsymbol{\phi}(\mathbf{z}), \boldsymbol{\phi}(\mathbf{z}) \rangle = K(\mathbf{x}, \mathbf{x}) K(\mathbf{z}, \mathbf{z}), \end{aligned}$$

These conditions are, however, not sufficient for the existence of a feature space, and we are in need of a more precise characterisation of kernels.

#### Mercer's Theorem

Mercer's theorem, which we will quote below, characterises when a bivariant function $K(\cdot, \cdot)$ on an input domain $X \subseteq \mathbb{R}^{n_x}$ is the inner product in some feature space $\mathcal{F}$. Before considering the case of a compact input domain, we will introduce the condition for $K(\cdot, \cdot)$ to be a kernel over a finite input space. As we will see, this condition, extended to any finite subspace of the input domain, is equivalent to Mercer's condition.

**Finite Case.** Suppose we are given a finite input space $X = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, and a symmetric function $K(\mathbf{x}, \mathbf{z})$ on $X$. We define the *kernel matrix*

$$\boldsymbol{K} = (K(\mathbf{x}_m, \mathbf{x}_n))_{m,n=1}^N, \quad K_{mn} = K(\mathbf{x}_m, \mathbf{x}_n),$$

which is symmetric. Therefore, we can write $\boldsymbol{K} = \boldsymbol{V}\boldsymbol{\Delta}\boldsymbol{V}^T$, with the diagonal matrix $\boldsymbol{\Delta}$ of eigenvalues $\lambda_j$ of $\boldsymbol{K}$, and the orthogonal matrix $\boldsymbol{V}$ of corresponding eigenvectors $\boldsymbol{v}_j = (v_{jn})_{n=1}^N$ as columns. What is more, we define a feature map $\boldsymbol{\phi} : \mathbf{x}_n \to \left(\sqrt{\frac{\lambda_j}{\gamma_j}} v_{jn}\right)_{j=1}^N$

into a feature space with an inner product, generalised by introducing a non-negative weighting $\gamma_j$ for each coordinate,

$$\langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z}) \rangle = \sum_{j=1}^{N} \gamma_j \phi_j(\mathbf{x}) \phi_j(\mathbf{z}).$$

With non-negative eigenvalues, the inner product between two feature vectors is

$$\begin{aligned}
\langle \boldsymbol{\phi}(\mathbf{x}_m), \boldsymbol{\phi}(\mathbf{x}_n) \rangle &= \left\langle \begin{bmatrix} \sqrt{\frac{\lambda_1}{\gamma_1}} v_{1m} \\ \vdots \\ \sqrt{\frac{\lambda_N}{\gamma_N}} v_{Nm} \end{bmatrix}, \begin{bmatrix} \sqrt{\frac{\lambda_1}{\gamma_1}} v_{1n} \\ \vdots \\ \sqrt{\frac{\lambda_N}{\gamma_N}} v_{Nn} \end{bmatrix} \right\rangle = \sum_{j=1}^{N} \lambda_j v_{jm} v_{jn} = \\
&= \left( [\boldsymbol{v}_1 \cdots \boldsymbol{v}_N] \begin{bmatrix} \lambda_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_N \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_N^T \end{bmatrix} \right)_{mn} = \\
&= \left( \boldsymbol{V} \boldsymbol{\Delta} \boldsymbol{V}^T \right)_{mn} = K_{mn} = K(\mathbf{x}_m, \mathbf{x}_n),
\end{aligned}$$

showing that $K(\cdot, \cdot)$ is the kernel function corresponding to the map $\boldsymbol{\phi}(\cdot)$. The eigenvalues have to be non-negative, as for a negative $\lambda_k$ with corresponding $\boldsymbol{v}_k$ a point

$$\begin{aligned}
\boldsymbol{\psi} = \sum_{n=1}^{N} v_{kn} \boldsymbol{\phi}(\mathbf{x}_n) &= \begin{bmatrix} \sqrt{\frac{\lambda_1}{\gamma_1}} v_{11} & \sqrt{\frac{\lambda_1}{\gamma_1}} v_{12} & \cdots & \sqrt{\frac{\lambda_1}{\gamma_1}} v_{1N} \\ \sqrt{\frac{\lambda_2}{\gamma_2}} v_{21} & \sqrt{\frac{\lambda_2}{\gamma_2}} v_{22} & \cdots & \sqrt{\frac{\lambda_2}{\gamma_2}} v_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ \sqrt{\frac{\lambda_N}{\gamma_N}} v_{N1} & \sqrt{\frac{\lambda_N}{\gamma_N}} v_{N2} & \cdots & \sqrt{\frac{\lambda_N}{\gamma_N}} v_{NN} \end{bmatrix} \begin{bmatrix} v_{k1} \\ \vdots \\ v_{kN} \end{bmatrix} = \\
&= \begin{bmatrix} \sqrt{\frac{\lambda_1}{\gamma_1}} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{\frac{\lambda_N}{\gamma_N}} \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_1^T \\ \vdots \\ \boldsymbol{v}_N^T \end{bmatrix} \boldsymbol{v}_k
\end{aligned}$$

has norm squared

$$\|\boldsymbol{\psi}\|^2 = \langle \boldsymbol{\psi}, \boldsymbol{\psi} \rangle = \boldsymbol{v}_k^T \boldsymbol{V} \sqrt{\boldsymbol{\Delta}} \sqrt{\boldsymbol{\Delta}} \boldsymbol{V}^T \boldsymbol{v}_k = \boldsymbol{v}_k^T \boldsymbol{V} \boldsymbol{\Delta} \boldsymbol{V}^T \boldsymbol{v}_k = \boldsymbol{v}_k^T \boldsymbol{K} \boldsymbol{v}_k = \lambda_k < 0.$$

This proves the following proposition.

**Proposition 23.** *Let $X$ be a finite input space, and $K(\cdot, \cdot)$ a symmetric function on $X$. Then $K(\cdot, \cdot)$ is a kernel function if and only if the matrix $\boldsymbol{K} = (K(\mathbf{x}_m, \mathbf{x}_n))_{m,n=1}^{N}$ is positive semi-definite [13].*

**Infinite Case.** For feature vectors $\boldsymbol{\phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), \cdots, \phi_j(\mathbf{x}), \cdots]$ Mercer's theorem gives the characterisation for a bounded, symmetric and continuous function $K(\mathbf{x}, \mathbf{z})$

to be a kernel[14],

$$K(\mathbf{x},\mathbf{z}) = \sum_{j=1}^{\infty} \gamma_j \phi_j(\mathbf{x})\phi_j(\mathbf{z}).$$

**Theorem 24 (Mercer).** *(e.g. in [13]) Let $X$ be a compact subset of $\mathbb{R}^{n_x}$, and $K$ a continuous symmetric function such that the integral operator $T_K$ is positive for all $f \in L_2(X)$,*

$$T_K : L_2(X) \to L_2(X),\ (T_K f)(\cdot) = \int_X K(\cdot,\mathbf{x}) f(\mathbf{x}) d\mathbf{x}$$

$$\int_{X\times X} K(\mathbf{x},\mathbf{z}) f(\mathbf{x}) f(\mathbf{z}) d\mathbf{x} d\mathbf{z} \geq 0.$$

*Then we can expand $K(\mathbf{x},\mathbf{z})$ in a uniformly convergent series on $X \times X$ in terms of the normalised eigenfunctions of $T_K$, $\phi_j \in L_2(X) : \|\phi_j\|_{L_2} = 1$, and associated positive eigenvalues $\lambda_j \geq 0$,*

$$K(\mathbf{x},\mathbf{z}) = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{x})\phi_j(\mathbf{z}).$$

The condition on a finite set of points $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ is recovered by choosing $f$ as the weighted sums of delta functions at the $\mathbf{x}_n$. Requiring the corresponding kernel matrix to be positive semi-definite for any finite subset on $X$ is therefore equivalent to Mercer's conditions, which is useful in practice for showing that a specific function is in fact a kernel.

The theorem suggests the feature mapping $\mathbf{x} \to \boldsymbol{\phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), \cdots, \phi_j(\mathbf{x}), \cdots]$ into the Hilbert space defined by the weighted inner product $\langle \boldsymbol{\theta}, \tilde{\boldsymbol{\theta}} \rangle = \sum_{j=1}^{\infty} \lambda_j \theta_j \tilde{\theta}_j$ with non-negative eigenvalues $\lambda_j$, since then

$$K(\mathbf{x},\mathbf{z}) = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{x})\phi_j(\mathbf{z}) = \langle \boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{z}) \rangle,$$

which is equivalent to $K(\mathbf{x},\mathbf{z})$ being an inner product in the feature space $\mathcal{F} \supseteq \boldsymbol{\phi}(X)$, where $\mathcal{F}$ is the $l_2$-space of all sequences $\boldsymbol{\psi} = [\psi_1, \cdots, \psi_j, \cdots]$ for which $\sum_{j=1}^{\infty} \lambda_j \psi_j^2 < \infty$. These features $\phi_j(\cdot)$ are orthonormal functions in $L_2(X)$ (*Mercer features*). The orthonormality condition is, however, not necessary. We can equivalently use a feature mapping with basis functions $\phi_j(\cdot)$ in which we rescale each coordinate, $\mathbf{x} \to \tilde{\boldsymbol{\phi}}(\mathbf{x}) = [a_1\phi_1(\mathbf{x}), \cdots, a_j\phi_j(\mathbf{x}), \cdots]$, into the Hilbert space defined by the weighted inner product $\langle \boldsymbol{\theta}, \tilde{\boldsymbol{\theta}} \rangle = \sum_{j=1}^{\infty} \frac{\lambda_j}{a_j^2} \theta_j \tilde{\theta}_j$. Then, the inner product of two feature vectors in $\mathcal{F}$ satisfies again $\langle \tilde{\boldsymbol{\phi}}(\mathbf{x}), \tilde{\boldsymbol{\phi}}(\mathbf{z}) \rangle = \sum_{j=1}^{\infty} \frac{\lambda_j}{a_j^2} a_j\phi_j(\mathbf{x}) a_j\phi_j(\mathbf{z}) = K(\mathbf{x},\mathbf{z})$. Orthogonality is not required either, as the case of the polynomial kernel shows, whose features are not in general orthogonal.

Mercer's theorem allows a representation of the input vectors $\mathbf{x}$ by means of their image in the feature space $\mathcal{F}$ with an inner product defined through the potentially infinite number of eigenvalues of the kernel. The sub-manifold formed by the image of the

[14]This contribution from functional analysis comes from studying the eigenvalue problem $\int K(\mathbf{x},\mathbf{z})\phi(\mathbf{z})d\mathbf{z} = \lambda\phi(\mathbf{x})$. For a function $K(\cdot,\cdot)$ to be bounded, it has to satisfy $\int\int K(\mathbf{x},\mathbf{z})d\mathbf{x}d\mathbf{z} < \infty$, a property that is also known as *finite trace*.

input space is defined through the eigenvectors of the kernel. The inner products can be calculated using the kernel function without computing the images $\boldsymbol{\phi}(\mathbf{x})$.

Up to this point, we have determined the properties a function $K(\cdot,\cdot)$ must have to be a kernel. Consider now a feature space with countably many linearly independent features $\mathbf{x} = [x_1, \cdots, x_N] \rightarrow \boldsymbol{\phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), \cdots, \phi_j(\mathbf{x}), \cdots]$, in the $l_2$-space $\mathcal{F}$ defined by the weighted inner product $\langle \boldsymbol{\theta}, \tilde{\boldsymbol{\theta}} \rangle = \sum_{j=1}^{\infty} \lambda_j \theta_j \tilde{\theta}_j$. We can define a function space $\mathcal{H}$ on the input space $X$ to be the image of $\mathcal{F}$ under the mapping

$$T : \boldsymbol{\theta} \rightarrow \sum_{j=1}^{\infty} \theta_j \phi_j(\mathbf{x}), \tag{2.53}$$

which for finite dimensional $\mathcal{F}$ corresponds to the function class we effectively use when applying linear functions in feature space, since it is the set of all linear combinations of the basis functions. For an $\mathcal{F}$ of infinite dimension, $\mathcal{H}$ may not contain all the possible hypothesis functions $h(\mathbf{x}) = \sum_{j=1}^{\infty} \theta_j \phi_j(\mathbf{x}) + b$, since they may be images of points that have an infinite norm in $\mathcal{F}$, or equivalently $\mathcal{H}$ may contain too many functions. In order to ensure that $\mathcal{H}$ contains exactly the set of hypothesis functions, we need to choose a particular feature map, which will be introduced in the next paragraph.

**Reproducing Kernel Hilbert Spaces (RKHS).** Assume we have a feature space given by the map $\mathbf{x} = [x_1, \cdots, x_N] \rightarrow \boldsymbol{\phi}(\mathbf{x}) = [\phi_1(\mathbf{x}), \cdots, \phi_j(\mathbf{x}), \cdots]$, in the $l_2$-space $\mathcal{F}$ defined by the weighted inner product

$$\langle \boldsymbol{\theta}, \tilde{\boldsymbol{\theta}} \rangle = \sum_{j=1}^{\infty} \lambda_j \theta_j \tilde{\theta}_j, \text{ with } K(\mathbf{x}, \mathbf{z}) = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{x}) \phi_j(\mathbf{z}).$$

Now consider introducing a weighting proportional to $\lambda_j$, so that the image of $\mathbf{x}$ in $\mathcal{F}$ is of form

$$\mathbf{x} \rightarrow \boldsymbol{\psi}(\mathbf{x}) = [\psi_1(\mathbf{x}), \cdots, \psi_j(\mathbf{x}), \cdots] = [\lambda_1 \phi_1(\mathbf{x}), \cdots, \lambda_j \phi_j(\mathbf{x}), \cdots], \tag{2.54}$$

with appropriate inner product

$$\langle \boldsymbol{\theta}, \tilde{\boldsymbol{\theta}} \rangle_{\lambda} = \sum_{j=1}^{\infty} \frac{\theta_j \tilde{\theta}_j}{\lambda_j}, \text{ such that } \langle \boldsymbol{\psi}(\mathbf{x}), \boldsymbol{\psi}(\mathbf{z}) \rangle_{\lambda} = K(\mathbf{x}, \mathbf{z}). \tag{2.55}$$

The image of $\mathcal{F}$ under the mapping (2.53) is $\mathcal{H}$. For two functions in $\mathcal{H}$

$$f(\mathbf{x}) = \sum_{j=1}^{\infty} \theta_j \phi_j(\mathbf{x}) \text{ and } g(\mathbf{x}) = \sum_{j=1}^{\infty} \tilde{\theta}_j \phi_j(\mathbf{x})$$

we define an inner product in $\mathcal{H}$,

$$\langle f(\cdot), g(\cdot) \rangle_{\mathcal{H}} = \sum_{j=1}^{\infty} \frac{\theta_j \tilde{\theta}_j}{\lambda_j},$$

making (2.53) an isometry. The particular weighting (2.54) has a couple of special properties. First, applying the map (2.53) to the image of an input point $\mathbf{z}$,

$$T(\boldsymbol{\psi}(\mathbf{z})) = \sum_{j=1}^{\infty} \psi_j(\mathbf{z})\phi_j(\mathbf{x}) = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{z})\phi_j(\mathbf{x}) = K(\mathbf{z}, \mathbf{x}),$$

shows that $K(\mathbf{z}, \cdot) \in \mathcal{H}$. Therefore, functions $h(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n K(\mathbf{x}_n, \mathbf{x})$ in dual representation are in $\mathcal{H}$. Next, taking the inner product of a general function $f(\mathbf{x}) = \sum_{j=1}^{\infty} a_j \phi_j(\mathbf{x}) \in \mathcal{H}$ with $K(\mathbf{z}, \cdot)$ gives

$$\langle f(\cdot), K(\mathbf{z}, \cdot)\rangle_{\mathcal{H}} = \sum_{j=1}^{\infty} \frac{a_j \lambda_j \phi_j(\mathbf{z})}{\lambda_j} = \sum_{j=1}^{\infty} a_j \phi_j(\mathbf{z}) = f(\mathbf{z})$$

(*reproducing property*), implying that $\mathcal{H}$ has to coincide with the closure of the subspace

$$\mathcal{K} = \left\{ \sum_{n=1}^{N} \alpha_n K(\mathbf{x}_n, \mathbf{x}) : (\mathbf{x}_1, \cdots, \mathbf{x}_N) \in X \right\},$$

since if

$$\begin{aligned} \forall f \in \mathcal{H}\text{: } & f \perp \mathcal{K} \\ & \Rightarrow f(\mathbf{z}) = \langle f(\cdot), K(\mathbf{z}, \cdot)\rangle_{\mathcal{H}} = 0,\ \forall \mathbf{z} \in X \\ & \Rightarrow f = 0. \end{aligned}$$

Thus, $\mathcal{H}$ does not contain functions that cannot be arbitrarily well approximated in the dual representation. Finally, the evaluation functionals defined by $F_{\mathbf{x}}(f) = f(\mathbf{x})\ \forall f \in \mathcal{H}$ are linear and bounded, since due to the reproducing property there exists $\beta = \|K(\mathbf{z}, \cdot)\|_{\mathcal{H}} \in \mathbb{R}^+$ such that[15]

$$|F_{\mathbf{z}}(f)| = |f(\mathbf{z})| = \langle f(\cdot), K(\mathbf{z}, \cdot)\rangle_{\mathcal{H}} \leq \|K(\mathbf{z}, \cdot)\|_{\mathcal{H}} \|f\|_{\mathcal{H}} \leq \beta \|f\|_{\mathcal{H}}\ \forall f \in \mathcal{H}.$$

These properties of the evaluation functionals of a Hilbert space $\mathcal{H}$ are exactly the defining properties of a *reproducing kernel Hilbert space* (RKHS). The importance of that is due to the Riesz representation theorem, which states that if $F_{(\cdot)}(\cdot)$ is a bounded linear functional on a Hilbert space $\mathcal{H}$ on $\mathcal{F}$, then there is a unique vector $\boldsymbol{\theta}$ in $\mathcal{F}$ such that $F_{\mathbf{x}}(f) = \langle f, \boldsymbol{\theta}\rangle_{\mathcal{H}}$.

**Theorem 25.** *For every Mercer kernel $K(\mathbf{x}, \mathbf{z})$ defined over a domain $X \subset \mathbb{R}^{n_x}$ there exists a RKHS $\mathcal{H}$ of functions defined over $X$ for which $K$ is the reproducing kernel (e.g. in [13]).*

The converse holds true as well, so that for any Hilbert space $\mathcal{H}$ of functions in which the evaluation functionals are linear and bounded, there exists a reproducing kernel. A reproducing kernel is also a Mercer kernel, since for $f(\mathbf{x}) = \sum_{n=1}^{N} \alpha_n K(\mathbf{x}_n, \mathbf{x})$,

$$\begin{aligned} 0 \leq \|f\|_{\mathcal{H}}^2 = \left\langle \sum_{n=1}^{N} \alpha_n K(\mathbf{x}_n, \cdot), \sum_{n=1}^{N} \alpha_n K(\mathbf{x}_n, \cdot) \right\rangle_{\mathcal{H}} = \sum_{m=1}^{N} \alpha_m \sum_{n=1}^{N} \alpha_n K(\mathbf{x}_m, \cdot) K(\mathbf{x}_n, \cdot) = \\ = \sum_{m=1}^{N} \sum_{n=1}^{N} \alpha_m \alpha_n K(\mathbf{x}_m, \mathbf{x}_n), \end{aligned}$$

[15] Remember that $K(\cdot, \cdot)$ has finite trace.

which proves that $K$ is positive semi-definite for every finite subset of $X$, implying the positivity of the operator[16] $K$.

Therefore, the set of hypothesis we actually employ when replacing the inner product in the support vector algorithms with a kernel $K(\cdot,\cdot)$ is the RKHS $\mathcal{H}$ defined by the inner product (2.55) in a space of features (2.54), with corresponding reproducing kernel $K(\cdot,\cdot)$.

### 2.5.4 Combining Kernels

We can construct more complicated kernels from simple building blocks. This is useful if one is in need of more complex kernels than the standard kernels, the Gaussian and the polynomial kernel, for a specific problem.

**Proposition 26.** *Let $K_1$ and $K_2$ be kernels over $X \times X$, $X \subseteq \mathbb{R}^{n_x}$, $a \in \mathbb{R}^+$, $f(\cdot)$ a real-valued function on $X$, $\phi \to \mathbb{R}^m$ with $K_3$ a kernel over $\mathbb{R}^m$, $\boldsymbol{B}$ a positive semi-definite $n \times n$-matrix, and $p(\cdot)$ a polynomial with positive coefficients. Then the following functions are kernels:*

1. $K(\mathbf{x},\mathbf{z}) = K_1(\mathbf{x},\mathbf{z}) + K_2(\mathbf{x},\mathbf{z})$,
2. $K(\mathbf{x},\mathbf{z}) = aK_1(\mathbf{x},\mathbf{z})$,
3. $K(\mathbf{x},\mathbf{z}) = K_1(\mathbf{x},\mathbf{z})K_2(\mathbf{x},\mathbf{z})$,
4. $K(\mathbf{x},\mathbf{z}) = f(\mathbf{x})f(\mathbf{z})$,
5. $K(\mathbf{x},\mathbf{z}) = K_3(\boldsymbol{\phi}(\mathbf{x}),\boldsymbol{\phi}(\mathbf{z}))$,
6. $K(\mathbf{x},\mathbf{z}) = \mathbf{x}^T\boldsymbol{B}\mathbf{x}$,
7. $K(\mathbf{x},\mathbf{z}) = p(K_1(\mathbf{x},\mathbf{z}))$,
8. $K(\mathbf{x},\mathbf{z}) = \exp(K_1(\mathbf{x},\mathbf{z}))$.

A proof for the rules quoted in the above proposition can be found in e.g. [13].

### 2.5.5 A Probabilistic View on SVMs

In this subsection, we want to establish a connection between SVMs and a particular form of Bayesian learning, known as a *Gaussian processes*. A Gaussian process is a stochastic process for which the marginal distribution for any finite set of variables is Gaussian. From now on we will, without loss of generality, assume all Gaussian distributions to be zero mean.

The output of a function $f(\mathbf{x})$ for fixed $\mathbf{x} \in X$, with $f$ chosen according to some distribution $\mathcal{D}$ over a class of real-valued functions $\mathcal{F}$, is a random variable. Therefore, $\{f(\mathbf{x}) : \mathbf{x} \in X\}$ is a collection of potentially correlated random variables, known as a stochastic process. The distribution $\mathcal{D}$ can be interpreted as the prior believe on how likely it is that a function $f$ will solve the problem under consideration. Such a prior

[16] A symmetric matrix $\mathbf{A}$ is positive semi-definite if and only if $\forall \mathbf{x} \neq 0 \Longrightarrow \mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$.

is a characteristic of Bayesian learning, in which one attempts to calculate the *posterior probability* of $\mathcal{F}$, $P(\mathcal{F}|S) = \frac{P(\mathcal{F})P(S|\mathcal{F})}{P(S)}$, given the data $S$. This is highly simplified by assuming the prior distribution to be Gaussian. For a finite data set $S = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$, the Gaussian distribution is specified by a symmetric positive definite covariance matrix $\boldsymbol{C}$, and defining such a prior distribution

$$P_{f\sim\mathcal{D}}\left(\{f(\mathbf{x}_1), \cdots, f(\mathbf{x}_N)\} = \{y_1, \cdots, y_N\}\right) \propto \exp(-\frac{1}{2}\mathbf{y}^T\boldsymbol{C}^{-1}\mathbf{y}) \tag{2.56}$$

corresponds to defining a Gaussian process on $\boldsymbol{f}(X)$[17]. The entries of the covariance matrix measure the correlation between $f(\mathbf{x}_m)$ and $f(\mathbf{x}_n)$, $\boldsymbol{C}_{mn} = E_{f\sim\mathcal{D}}\{f(\mathbf{x}_m)f(\mathbf{x}_n)\}$, which depends only on $\mathbf{x}_m$ and $\mathbf{x}_n$. Thus, a symmetric covariance function $K(\mathbf{x}, \mathbf{z})$ exists such that $\boldsymbol{C}_{mn} = K(\mathbf{x}_m, \mathbf{x}_n)$. The covariance matrix has to be positive definite for all finite sets of input points, which is exactly the property defining a Mercer kernel, and hence defining a Gaussian process over a set of variables on $X$ is equivalent to defining a Mercer kernel on $X \times X$ [75].

Usually, a Gaussian process is defined by specifying a covariance function $C(\mathbf{x}, \mathbf{z})$ as this avoids the explicit definition of the function class $\mathcal{F}$. It is possible to define a function class and prior for which the kernel is the corresponding covariance function. Consider the class of linear functions in the space $\mathcal{F}$ of Mercer features $\mathbf{x} \to \boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \cdots, \phi_j(\mathbf{x}), \cdots)$ in the $L_2$ space defined by the weighted inner product $\langle\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}\rangle = \sum_{j=1}^{\infty}\lambda_j\theta_j\tilde{\theta}_j$,

$$f(\mathbf{x}) = \sum_{j=1}^{\infty}\theta_j\phi_j(\mathbf{x}) = \sum_{n=1}^{N}\alpha_n K(\mathbf{x}_n, \mathbf{x}). \tag{2.57}$$

By choosing the prior $\mathcal{D}$ on the weight vector $\boldsymbol{\theta}$ as an independent zero mean Gaussian in each coordinate, with variances $\sqrt{\lambda_j}$, the correlation $C(\mathbf{x}, \mathbf{z})$ between $\mathbf{x}$ and $\mathbf{z}$ can be computed [13]:

$$\begin{aligned} C(\mathbf{x}, \mathbf{z}) &= E_{\boldsymbol{\theta}\sim\mathcal{D}}\left\{f(\mathbf{x})f(\mathbf{z})\right\} = E_{\boldsymbol{\theta}\sim\mathcal{D}}\left\{\sum_{j=1}^{\infty}\theta_j\phi_j(\mathbf{x})\sum_{j=1}^{\infty}\theta_j\phi_j(\mathbf{z})\right\} = \\ &= \int_{\mathcal{F}}\sum_{j=1}^{\infty}\theta_j\phi_j(\mathbf{x})\sum_{j=1}^{\infty}\theta_j\phi_j(\mathbf{z})d\mathcal{D}(\boldsymbol{\theta}) = \sum_{i=1}^{\infty}\sum_{j=1}^{\infty}\phi_i(\mathbf{x})\phi_j(\mathbf{z})\int_{\mathcal{F}}\theta_i\theta_j d\mathcal{D}(\boldsymbol{\theta}) = \\ &= \sum_{i=1}^{\infty}\sum_{j=1}^{\infty}\phi_i(\mathbf{x})\phi_j(\mathbf{z})\delta_{ij}\lambda_j = \sum_{i=j}^{\infty}\phi_j(\mathbf{x})\phi_j(\mathbf{z})\lambda_j = K(\mathbf{x}, \mathbf{z}). \end{aligned}$$

Thus, defining a Gaussian prior on the weights $\boldsymbol{\theta}$ of (2.57), independent and with variance $\sqrt{\lambda_j}$ in each coordinate, is equivalent to defining a prior distribution (2.56) with covariance matrix $\boldsymbol{C}_{mn} = K(\mathbf{x}_m, \mathbf{x}_n)$ on $\boldsymbol{f}(X)$.

Now consider again the class of linear functions (2.57) in the space $\mathcal{F}$ of Mercer features as above, in the more specific case when $S = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$ is assumed to be generated i.i.d. according to

$$P(\mathbf{x}, y) = P(y|\mathbf{x})P(\mathbf{x}),$$

[17]We write $\boldsymbol{f}(X) = [f(\mathbf{x}_1), \cdots, f(\mathbf{x}_N)]$ for the column vector of the output values of $f$ at the points $\mathbf{x}_n$.

with noise distribution

$$P(y|\mathbf{x}) = P(y - f(\mathbf{x})) \propto \exp(-CL(y - f(\mathbf{x}))). \tag{2.58}$$

The likelihood of $f$ is then

$$P(S|\boldsymbol{f}(X)) = \exp\left(-C\sum_{n=1}^{N} L(y_n - f(\mathbf{x}_n))\right).$$

By defining the prior $P_{f\sim\mathcal{D}}$ as a Gaussian process (2.56), with covariance matrix $\boldsymbol{K}_{X\times X} = \{K(\mathbf{x}_m, \mathbf{x}_n)\}_{m,n=1}^{N}$, where $K(\cdot,\cdot)$ is a kernel as discussed above, the posterior distribution takes the form

$$P(\boldsymbol{f}(X)|S) = \frac{P(S|\boldsymbol{f}(X))P(\boldsymbol{f}(X))}{P(S)} = \\ \propto \exp\left(-\frac{1}{2}\boldsymbol{f}^T(X)\boldsymbol{K}_{X\times X}^{-1}\boldsymbol{f}(X) - C\sum_{n=1}^{N} L(y_n - f(\mathbf{x}_n))\right).$$

The maximum a posterior estimate for $f(\mathbf{x})$ is $f(\mathbf{x})_{\text{MAP}} = \arg\max_{f(\mathbf{x})} P(\boldsymbol{f}(X)|S)$. Equivalently, we can use the log-posterior,

$$f(\mathbf{x})_{\text{MAP}} = \arg\min_{f(\mathbf{x})} \frac{1}{2}\boldsymbol{f}^T(X)\boldsymbol{K}_{X\times X}^{-1}\boldsymbol{f}(X) + C\sum_{n=1}^{N} L(y_n - f(\mathbf{x}_n)). \tag{2.59}$$

By substituting for our latent functions $f$ (2.57),

$$\boldsymbol{f}(X) = \begin{bmatrix} \sum_{n=1}^{N}\alpha_n K(\mathbf{x}_n, \mathbf{x}_1) \\ \vdots \\ \sum_{n=1}^{N}\alpha_n K(\mathbf{x}_n, \mathbf{x}_N) \end{bmatrix} = \sum_{n=1}^{N}\alpha_n \begin{bmatrix} K(\mathbf{x}_n, \mathbf{x}_1) \\ \vdots \\ K(\mathbf{x}_n, \mathbf{x}_N) \end{bmatrix} = \boldsymbol{K}_{X\times X}\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix}, \tag{2.60}$$

and substituting (2.60) back into (2.59), the maximum a posterior estimate for $f(\mathbf{x})$ becomes

$$f(\mathbf{x})_{\text{MAP}} = \arg\min_{f(\mathbf{x})} \frac{1}{2}\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix}^T \boldsymbol{K}_{X\times X}\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix} + C\sum_{n=1}^{N} L(y_n - f(\mathbf{x}_n)).$$

This corresponds exactly to the type of criterion that is optimised in all the SVM algorithms, which is of form

$$\|f\|_{\mathcal{H}}^2 + C\sum_{n=1}^{N} L(y_n - f(\mathbf{x}_n)),$$

since the norm $\|f\|_{\mathcal{H}}^2$ can be expressed as

$$\|f\|_{\mathcal{H}}^2 = \left\langle \sum_{m=1}^{N}\alpha_m K(\mathbf{x}_m, \mathbf{x}), \sum_{n=1}^{N}\alpha_n K(\mathbf{x}_n, \mathbf{x}) \right\rangle = \sum_{m=1}^{N}\sum_{n=1}^{N}\alpha_m\alpha_n \langle K(\mathbf{x}_m, \mathbf{x}), K(\mathbf{x}_n, \mathbf{x})\rangle = \\ = \sum_{m=1}^{N}\sum_{n=1}^{N}\alpha_m\alpha_n K(\mathbf{x}_m, \mathbf{x}_n) = \begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix}^T \boldsymbol{K}_{X\times X}\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_N \end{bmatrix}.$$

Therefore, the solution of the support vector algorithms is equivalent to the MAP estimate by a Gaussian process with covariance function $K(\cdot,\cdot)$, when a noise distribution of type (2.58), with $L(\cdot)$ a loss function as discussed in subsection 2.3.10, is assumed.

## 2.6 Hyperparameter Selection

SVMs have the advantage over other function estimation methods, e.g. classical neural networks, that they are capable of controlling the capacity of the hypothesis, even when the hypothesis space is very rich. The algorithm chooses the subspace of the hypothesis space that is optimal in terms of some bound on the generalisation of the hypothesis. The "architecture" of the learning machine is therefore determined by the learning algorithm itself. This leaves only very few parameters to be chosen by the user. Concretely, the user has to determine the value of the trade off parameter $C$, the width of the $\epsilon$-insensitive zone or the corresponding parameter $\nu$, and the kernel parameter(s). We write these $J$ *hyperparameters* $\Theta_j$ jointly as a parameter vector $\mathbf{\Theta} = [\Theta_j]_{j=1}^{J} \in \mathbf{\Gamma}$, where $\mathbf{\Gamma}$ is the *parameter space*.

Up to this point of the work, the question of how to choose these parameters $\mathbf{\Theta}$ has been left unanswered. The different possibilities for determining them will be summarised in this section.

It is known that the excellent empirical performance of SVMs depends crucially on a good choice of hyperparameters. Although several different approaches exist for their selection, the problem of how to practically choose a good set is still far from being solved.

### 2.6.1 Generalisation Estimation and Parameter Space Search

Hyperparameter selection methods consist of two building blocks. The first block estimates the performance (generalisation) of the learning machine for a specific parameter setting $\mathbf{\Theta}_i$, given only an available set of $N$ samples $S = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}$. The second block determines how such candidate settings $\mathbf{\Theta}_i$ are selected in parameter space $\mathbf{\Gamma}$.

#### Generalisation Estimators

We distinguish three approaches:

- *Empirical methods* approximate the generalisation by evaluation of the hypotheses on dedicated subsets $S_{\text{valid}} \subset S$ of the available set of samples $S$. For each *validation set* $S_{\text{valid}}$ and parameter setting $\mathbf{\Theta}_i$, the hypothesis[18] $h_{\mathbf{\Theta}_i}(S_{\text{train}} = S \setminus S_{\text{valid}})$ has to be known[19].

- Methods based on *bounds on the risk* approximate the generalisation by the value of a bound. The bounds depend on properties of the available set of samples $S$ and

[18] In this section, we write shortly $h_{\mathbf{\Theta}_i}(S)$ for the hypothesis that is obtained through training on the training set $S$ with hyperparameters $\mathbf{\Theta}_i$.

[19] In the case of SVMs, finding the hypothesis $h_{\mathbf{\Theta}_i}(S_j)$ means running the support vector training algorithm for this specific parameter setting and training set.

of the hypothesis $h_{\boldsymbol{\Theta}_i}(S)$. The method therefore requires the computation of $h_{\boldsymbol{\Theta}_i}(S)$ for each parameter setting $\boldsymbol{\Theta}_i$ considered.

- *Heuristic methods* aim at finding an empirical functional relationship between $\boldsymbol{\Theta}_i$ and the generalisation of its corresponding hypothesis $h_{\boldsymbol{\Theta}_i}(S)$. The functional relationship is parameterised only by properties of the available set of samples $S$ and does not depend on properties of the specific hypotheses $h_{\boldsymbol{\Theta}_i}(S)$. Therefore, it is not necessary to compute $h_{\boldsymbol{\Theta}_i}(S)$ for each $\boldsymbol{\Theta}_i$. Heuristics are usually only valid within a particular problem setting.
  A heuristic method for SVR is reported in [11]. We could, however, not produce good results with this method in own experiments. We will not consider heuristics any further.

#### Searching the Parameter Space

Both empirical methods and methods based on bounds require the hypothesis to be known for each specific parameter setting $\boldsymbol{\Theta}_i$ under consideration. For that, candidate vectors $\boldsymbol{\Theta}_i$ have to be selected in $\boldsymbol{\Gamma}$. The number of candidate vectors $\boldsymbol{\Theta}_i$ largely determines the computational requirements of the hyperparameter selection procedures, since the hypothesis has to be computed for each specific hyperparameter setting and validation set.

**Grid Search.** A common brute-force approach towards selecting candidate hyperparameter vectors $\boldsymbol{\Theta}_i$ is known as a *grid search*. For each of the $J$ hyperparameter $\Theta_j$, a discrete set of $n_j$ values $\Theta_j = [\Theta_{j,1}, \cdots, \Theta_{j,n_j}]$ is selected, and every possible combination of these hyperparameter values is considered. This places a finitely sampled grid supported by $\prod_{j=1}^{J} n_j \approx \bar{n}^J$ vectors $\boldsymbol{\Theta}_i$ in the parameter space. For each $\boldsymbol{\Theta}_i$, the hypothesis $h_{\boldsymbol{\Theta}_i}(S)$ has to be evaluated.

**Gradient Descent Techniques.** If the generalisation estimator $\mathrm{Err}(\boldsymbol{\Theta}, S)$ is differentiable with respect to the hyperparameters $\Theta_j$, gradient descent type methods can be used. Starting from an initial hyperparameter vector $\boldsymbol{\Theta}_1$, the next vector $\boldsymbol{\Theta}_2$ is chosen to be in the direction of steepest descent $-\frac{\partial \mathrm{Err}(\boldsymbol{\Theta}, S)}{\partial \boldsymbol{\Theta}}$. The procedure is iterated until some termination criterion is fulfilled. For each $\boldsymbol{\Theta}_i$, the hypothesis $h_{\boldsymbol{\Theta}_i}(S)$ has to be evaluated.

### 2.6.2 Empirical Generalisation Estimators

All empirical methods have in common that the available set of samples $S$ is split into training sets and validation sets $S_{i,\mathrm{train}}, S_{i,\mathrm{valid}}$: $S_{i,\mathrm{train}} \cup S_{i,\mathrm{valid}} = S$, $S_{i,\mathrm{train}} \cap S_{i,\mathrm{valid}} = \{\}$. The hypotheses $h_{\boldsymbol{\Theta}_i}(S_{\mathrm{train}})$ are determined on the training sets $S_{\mathrm{train}}$, and the validation sets $S_{\mathrm{valid}}$ are sacrificed for estimating the generalisation.

Empirical methods can be used for performance evaluation of any function estimation technique. They are suitable for all types of SVMs we have considered.

Empirical generalisation estimators $\mathrm{Err}(\boldsymbol{\Theta}, S)$ are not differentiable with respect to $\Theta_j$. Therefore, gradient descent techniques are not applicable, and a grid search has to

be performed. Thus, they are very costly in terms of computation time, although there exist strategies to speed up the procedures [8].

In general, empirical generalisation estimators produce good, nearly unbiased estimates of the generalisation error [10] [15]. We distinguish between *hold out testing*, *k-fold cross validation* and *leave one out* estimators.

**Hold Out Testing.** The available set of samples $S$ of length $N$ is divided into a single training set and a single validation set of length $m$ and $n$, respectively, such that

$$S_{\text{train}}, S_{\text{valid}}\text{: } S_{\text{train}} \cup S_{\text{valid}} = S, S_{\text{train}} \cap S_{\text{valid}} = \{\}.$$

For each choice of hyperparameters $\boldsymbol{\Theta}_i$, the machine is trained on $S_{\text{train}}$, and the generalisation is estimated on $S_{\text{valid}}$ by the estimator

$$\text{Err}_{\text{HO}}^{(m,n)}\left(h_{\boldsymbol{\Theta}_i}\right) = \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in S_{\text{test}}} L\left(h_{\boldsymbol{\Theta}_i}(\mathbf{x}_i, y_i)\right).$$

The procedure is repeated for all other choices of hyperparameters $\boldsymbol{\Theta}_i \in \boldsymbol{\Gamma}$, and the hyperparameters $\boldsymbol{\Theta}^*$ with the best estimated generalisation, $\forall i$: $\text{Err}_{\text{HO}}^{(m,n)}\left(h_{\boldsymbol{\Theta}^*}\right) \leq \text{Err}_{\text{HO}}^{(m,n)}\left(h_{\boldsymbol{\Theta}_i}\right)$, is selected.

**k-Fold Cross Validation.** The available set of samples $S$ is split into $k$ mutually exclusive subsets $S_i$ of approximately equal size $n$, $S_i$: $\bigcup_{i=1}^{k} S_i = S$, $\bigcap_{i=1}^{k} S_i = \{\}$. For each particular choice of hyperparameters $\boldsymbol{\Theta}_i$, the machine is trained on the training sets $S_{j,\text{train}} = \bigcup_{i \neq j} S_i$, $j = 1, \cdots, k$, and then tested on the corresponding validation sets $S_{j,\text{valid}} = S_j$, $j = 1, \cdots, k$, consisting of the subset that has not been used for training. The estimated generalisation for a specific hyperparameter setting $\boldsymbol{\Theta}_i$ is then the average of the validation error on the validation sets $S_{j,\text{valid}}$. Therefore, the machine has to be trained $k$ times for each candidate $\boldsymbol{\Theta}_i$. The estimator is given by

$$\text{Err}_{\text{kCV}}^{(n)}\left(h_{j\boldsymbol{\Theta}_i}\right) = \frac{1}{k} \sum_{j=1}^{k} \left( \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in S_{j,\text{valid}}} L\left(h_{j\boldsymbol{\Theta}_i}(\mathbf{x}_i, y_i)\right) \right).$$

The procedure is repeated for all other choices of hyperparameters $\boldsymbol{\Theta}_i \in \boldsymbol{\Gamma}$, and the hyperparameter setting $\boldsymbol{\Theta}^*$ with the best estimated generalisation, $\forall i$: $\text{Err}_{\text{kCV}}^{(n)}\left(h_{\boldsymbol{\Theta}^*}\right) \leq \text{Err}_{\text{kCV}}^{(n)}\left(h_{\boldsymbol{\Theta}_i}\right)$, is selected.

k-fold cross validation gives better estimates of the generalisation than simple hold out testing, at the expense of approximately $k$ times the computational cost [10] [15].

**Leave One Out Estimator.** The leave one out estimator can be seen as an extreme form of the k-fold cross validation estimator for which $k$ is equal to the number of available samples $N$. Thus, only one sample is left out for testing, and training and testing has to be repeated $N$ times for each particular setting of hyperparameters $\boldsymbol{\Theta}_i$. The estimator is given by

$$\text{Err}_{\text{LOO}}^{(1)}\left(h_{n\boldsymbol{\Theta}_i}\right) = \frac{1}{N} \sum_{n=1}^{N} L\left(h_{n\boldsymbol{\Theta}_i}(\mathbf{x}_n, y_n)\right).$$

The procedure is repeated for all other choices of hyperparameters $\mathbf{\Theta}_i \in \mathbf{\Gamma}$, and the hyperparameter setting with the best estimated generalisation is selected.

The leave one out estimator is known to give the best estimate on the generalisation of all empirical methods, at the expense of approximately $N$ and $\frac{N}{k}$ times the computational cost of hold out testing and k-fold cross validation, respectively [10] [15].

### 2.6.3 Estimators based on Bounds on the Generalisation

Instead of using empirical estimates of the generalisation performance, for which a partition of the training data has to be dedicated to validation, we can try to approximate the generalisation performance by theoretical bounds. Then, for each choice of hyperparameters $\mathbf{\Theta}_i \in \mathbf{\Gamma}$, the performance is assessed via the value of the bound. This is computationally less costly than empirical estimates if the bounds are differentiable with respect to the hyperparameters $\Theta_j$ and a gradient descent technique can be employed, since then the number of tested hyperparameter settings $\mathbf{\Theta}_i$ may be considerably lower. Reliable and robust generalisation bounds for hyperparameter selection are still subject to active research, which mainly focuses on bounds for classifiers. We will therefore not go further into details and leave the reader with selected examples and corresponding pointers to relevant literature.

The *xi-alpha bound* is an upper bound on the leave one out estimate and can be computed without extra cost when training support vector classifiers [15] [30]. Other bounds for classification are the *generalised approximate cross validation* (GACV) and the *span bound*. The GACV approximates the generalised comparative Kullback-Liebler distance, which is seen as an upper bound on the misclassification rate, and can be computed directly once the SVM is trained on the whole training set [36]. The span bound approximates the leave one out estimate [10] [70].

All bounds introduced in Section 2.2 and 2.3 can be used to approximate the generalisation [5]. As this bounds are considerably loose, they may, however, not produce usable estimates [15].

### 2.6.4 Practical Considerations

Cross validation and leave one out estimates are known to produce good estimates of the generalisation, but they are practically not feasible for tuning many parameters (e.g. when using kernels with multiple parameters) due to the computational cost of these techniques.

Computationally less expensive methods are still subject to research and discussion. Past efforts focused on finding tight bounds on the leave one out estimate (e.g. xi-alpha bound, span bound), but they are not satisfactory in many situations. A comparative study of inexpensive generalisation measures for tuning the trade off parameter $C$ and the width of a Gaussian kernel in the case of support vector classification can be found in [15], for a variety of data sets. It is observed there that the xi-alpha bound gives hyperparameters that are sometimes not close to the optimal ones, GACV is worse than the xi-alpha bound on some data sets, and the VC bound (2.8) does not give useful estimates at all.

Until reliable bounds and iterative techniques are found, empirical techniques depending on validation sets will be the methods of choice for selecting the hyperparameters of SVMs. This shortcoming of reliable hyperparameter selection techniques clearly restricts the power of SVMs in practice:

1. Due to the computational cost of all empirical techniques, the parameter space cannot be sampled arbitrarily fine when performing the necessary grid search. Therefore, it cannot be guaranteed that the hyperparameter setting selected is optimal for a particular problem.

2. The computational cost of a grid search is approximately proportional to $n^J$, where $n$ is the size of the set of discrete values considered for one single hyperparameter $\Theta_j$, and $J$ is the number of hyperparameters involved. Therefore, the computational cost depends crucially on the number of hyperparameters $J$. This limits SVMs to the use of kernels with only one or very few kernel parameters.

In the practical experiments reported in Chapter 4 of this work, we use simple hold out testing on a single validation set. Although this is the computationally least costly empirical parameter selection technique, it leaves us restricted to considering only standard kernels of Gaussian and polynomial type (2.52) and (2.51).

# Chapter 3

# Application to Chaotic Time Series Prediction

## 3.1 Chaos and Dynamical Systems

A variety of experiments have shown that a measured time series can in many cases be viewed as driven by a nonlinear *deterministic dynamical system* with low dimensional *chaotic attractor* [14].

A deterministic dynamical systems describes the time evolution of a *state* $\boldsymbol{x}$ in some *phase space* or *state space* $\mathcal{P} \subseteq \mathbb{R}^D$. The system can be expressed by ordinary differential equations or, in discrete time , by maps of the form

$$\boldsymbol{x}(n+1) = \boldsymbol{f}(\boldsymbol{x}(n)), \tag{3.1}$$

where $\boldsymbol{x}(n)$ are the states of the system at time instance $n$. The time evolution of these states in the phase space $\mathcal{P}$ is known as an *orbit* or *trajectory*.

The state $\boldsymbol{x}(n)$ of a deterministic dynamical system gives all the information necessary to determine the entire future evolution of the system. Under minimal requirements on the smoothness of the equation (3.1), the existence and uniqueness properties hold. They assure that through any point $\boldsymbol{x}(n) = (x_1(n), \cdots, x_D(n)) \in \mathcal{P}$, there exists a unique trajectory with $\boldsymbol{x}(n)$ as the initial condition. Therefore, the future of the system that unfolds from the state $\boldsymbol{x}(n)$ is uniquely determined by $\boldsymbol{x}(n)$ [50].

Chaos can occur as a feature of such orbits $\boldsymbol{x}(n)$, if they are arising from nonlinear evolution rules which are either systems of differential equations with three or more degrees of freedom, or of invertible discrete time maps (3.1) with two or more degrees of freedom, or of noninvertible maps in one or more dimensions. The degrees of freedom of a system of differential equations is the number of required first order ordinary differential equations, and the degrees of freedom of invertible discrete time maps (3.1) is the number of components in the state vector $\boldsymbol{x}(n)$ [1]. Chaos manifests itself as complex time traces of the state variables $\boldsymbol{x}(n)$, with continuous, broad band Fourier spectra and non-periodic motion. As a class of observable signals, chaos is logically situated in between the domain of predictable, regular, or quasi-periodic signals, and signals with truly stochastic behaviour, which are completely unpredictable. With conventional linear

tools such as Fourier transforms, chaos looks like "noise". This is because frequency domain methods lack applicability, as chaotic signals typically possess a continuous Fourier spectrum, whereas time domain methods become inappropriate because a single model no longer applies to the entire state space underlying the signal. Nevertheless, chaos has structure in an appropriate state or phase space [1] [50].

A characteristic feature of deterministic chaotic systems is an extreme, exponential sensitivity to changes in initial conditions, while the dynamics is still constrained to a finite region of the state space, called a strange attractor. This sensitivity places a fundamental limit on long-term predictions. For given initial conditions with finite precision, the long term behaviour cannot be predicted, except to say that the states are constrained to a certain finite region of the state space. Initially nearby trajectories may diverge exponentially for chaotic systems[1]. Still, the determinism of the system suggests possibilities for short-term prediction: Random-looking data may contain simple relationships, involving only a few irreducible degrees of freedom [18] [32].

**Observed Chaos.** Chaotic behaviour is usually observed as a property of measurements in form of a *time series*. A time series can be thought of as a (usually scalar) sequence of observations

$$\{s(n) = \Omega(\boldsymbol{x}(n)\}, \quad n = 1, \cdots, N, \tag{3.2}$$

performed with some measuring function $\Omega(\cdot)$. When we observe a time series (3.2), the system $\boldsymbol{f}$, its attractor dimension[2] $d_A$, and the measurement function $\Omega$ are unknown. We possess only knowledge of the measurements themselves and of the sampling time[3] $t_s$.

## 3.2 Predicting Chaotic Time Series

In many situations in science and technology we face the necessity of predicting the future evolution a time series (3.2) of measurements of the state of a system from its past observations. The classical approach is to build an explanatory model from assumptions on the system and measure initial data. Mathematical models of the system are then investigated by writing down the equations of motion and by trying to integrate them forward in time, to predict the future state of the system. Unfortunately, this is not always possible, due to a lack of knowledge of the system and/or a lack of initial data [18] [43].

Another approach to prediction is to identify the present state of the system which is producing the time series, and to use the available history of states to estimate its future evolution. By studying the evolution of the observable following similar states, information about the future can be inferred. Due to the existence and uniqueness properties of deterministic dynamical systems, the present state $\boldsymbol{x}(n)$ contains all information needed to produce the state $\tau_p$ time steps in the future,

$$\boldsymbol{x}(n + \tau_p) = \boldsymbol{f}_{\tau_p}(\boldsymbol{x}(n)).$$

[1] The rate of the exponential divergence of trajectories is measured by the *Lyapunov exponent*.

[2] For definitions of dimension measures for chaotic systems, for instance the box counting dimension and the Lyapunov dimension, see e.g. [1]

[3] The sampling time $t_s$ is connecting the discrete time $n$ and the continuous time $t$ through $t = t_0 + nt_s$, where $t_0$ is some constant offset.

When taking this approach, the prediction problem naturally breaks down into two subproblems: a representation problem and a function approximation problem.

In the representation problem, we attempt to reconstruct the present state from the scalar time series (3.2) or, in other words, to use the available data to find a vector $\boldsymbol{y}(n)$ with which to replace the theoretical state $\boldsymbol{x}(n)$. Since the available sequence of observations $\{s(n)\}$ in itself does not properly represent the (multi-dimensional) phase space of the underlying dynamical system, we have to employ some technique to unfold this multi-dimensional structure. We have to seek to reconstruct an attractor from the available scalar data in an embedding space that preserves the invariant characteristics of the original unknown attractor of $\boldsymbol{f}$. A particularly elegant solution to the problem is *delay coordinate embedding*, where each state $\boldsymbol{x}(n)$ is identified with a unique vector $\boldsymbol{y}(n)$ in an Euclidean space $\mathbb{R}^m$ [63]. The *embedding dimension* $m$ will in general be different from the unknown dimension $\lfloor d_A \rfloor$ [26] [32]. This approach will be made clear in the next subsection.

Once we found a proper representation $\boldsymbol{y}(n)$ for the theoretical state $\boldsymbol{x}(n)$ of the system, the problem remaining is to find an efficient approximation $\hat{P}_{\tau_p}$ for the prediction function $P_{\tau_p}$,

$$P_{\tau_p} : \mathbb{R}^m \mapsto \mathbb{R} \tag{3.3}$$

$$\hat{P}_{\tau_p} : \mathbb{R}^m \mapsto \mathbb{R} \tag{3.4}$$

$$P_{\tau_p}(\boldsymbol{y}(n)) = s(n+\tau_p) = \Omega(\boldsymbol{x}(n+\tau_p)) = \Omega(\boldsymbol{f}_{\tau_p}(\boldsymbol{x}(n))) \tag{3.5}$$

$$\hat{P}_{\tau_p}(\boldsymbol{y}(n)) = \hat{s}(n+\tau_p) \approx P_{\tau_p}(\boldsymbol{y}(n)) = s(n+\tau_p), \tag{3.6}$$

using the available data. If a good approximation can be found, prediction involves locating the present state vector $\boldsymbol{y}(n)$, and evaluating $\hat{P}_{\tau_p}$ [50].

### 3.2.1 Representation Problem

The answer to the question how to go from scalar observations (3.2) to a multivariate phase space is contained in Takens' embedding theorem [63] and its refinement by Sauer et al. [51], leading to the time delay coordinate embedding.

The theorem tells us that if we are able to observe a scalar quantity $\Omega(\cdot)$ of some vector function $\boldsymbol{g}(\boldsymbol{x}(n))$ of the dynamical variables of a deterministic dynamical system

$$\boldsymbol{x}(n) \mapsto \boldsymbol{f}_{\tau_p}(\boldsymbol{x}(n)) = \boldsymbol{x}(n+\tau_p),$$

then the geometric structure of the multivariate system can be unfolded from this set of scalar measurements $\Omega(\boldsymbol{g}(\boldsymbol{x}(n)))$, in a space made out of new vectors with components consisting of $\Omega(\cdot)$ applied to powers of $\boldsymbol{g}(\boldsymbol{x}(n))$. These vectors

$$\boldsymbol{y}(n) = [\Omega(\boldsymbol{x}(n)), \Omega(\boldsymbol{g}^{\tau}(\boldsymbol{x}(n))), \Omega(\boldsymbol{g}^{2\tau}(\boldsymbol{x}(n))), \cdots, \Omega(\boldsymbol{g}^{(m-1)\tau}(\boldsymbol{x}(n)))]$$

define motion in an $m$-dimensional Euclidean space. With quite general conditions of smoothness on the functions $\Omega(\cdot)$ and $\boldsymbol{g}(\boldsymbol{x})$, it is shown that if $m$ is large enough, then many important properties of the unknown multivariate signal $\boldsymbol{x}(n)$ at the source of the observed chaos are reproduced without ambiguity in the new space of vectors $\boldsymbol{y}(n)$. In

particular, it is shown that the sequential order of the points $\boldsymbol{y}(n) \rightarrow \boldsymbol{y}(n+\tau_p)$, the evolution in time, follows that of the unknown dynamics $\boldsymbol{x}(n) \rightarrow \boldsymbol{x}(n+\tau_p)$. In other words, the state $\boldsymbol{x}(n)$ of the deterministic dynamical system, and therefore its future evolution, is completely specified by its corresponding vector $\boldsymbol{y}(n)$ [1] [50].

Any smooth choice for $\Omega(\cdot)$ and $\boldsymbol{g}(\boldsymbol{x})$ is possible and we can, in particular, use for the general function $\boldsymbol{g}(\boldsymbol{x})$ the operation which takes some initial vector $\boldsymbol{x}$ to the vector time delays $\delta = \tau \cdot t_s$ before. Then the vectors

$$\boldsymbol{y}(n) = [s(n), s(n-\tau), s(n-2\tau), \cdots, s(n-(m-1)\tau)],$$

are simply composed of the observations at time lags $\tau$. These *time delay vectors* $\boldsymbol{y}(n) \in \mathcal{R} \subseteq \mathbb{R}^m$ replace the scalar measurements $s(n)$ with vectors in an $m$-dimensional Euclidean space in which the invariant aspects of the sequence of points $\boldsymbol{x}(n)$ are captured with no loss of information about the properties of the original system. The new space $\mathcal{R}$ is related to the original space $\mathcal{P}$ by smooth, differentiable transformations. This means that we can work in the reconstructed time delay space $\mathcal{R}$ and learn essentially as much about the system at the source of our observations as we could if we were able to make our calculations directly in the "true" space $\mathcal{P}$ in which the true states $\boldsymbol{x}(n)$ live. What time lag $\tau$ and what embedding dimension $m$ to use are the central issues of this reconstruction [1].

Takens proved that for an infinite amount of noise free data it is sufficient that

$$m \geq 2d_A + 1,$$

where $d_A$ is the box counting dimension of the attractor of $\boldsymbol{f}$. The theorem guarantees that the attractor embedded in the $m$-dimensional state space $\mathcal{R}$ is "unfolded" without any self-intersections. Under the idealized conditions of the theorem, the actual values of $\tau$ and the sampling interval $t_s$ of the discrete time system are irrelevant; in practice, however, this is not the case.

The condition $m \geq 2d_A + 1$ is sufficient but not necessary, and an attractor may in practice be successfully reconstructed with an embedding dimension as low as $\lfloor d_A \rfloor + 1$ [32]. We will investigate methods for determining prescriptions for the embedding dimension $m$ and the delay $\tau$ directly from the observations in the next subsection, after a geometric interpretation of Takens' Theorem.

**Geometric Interpretation of Embedding.** The basic idea of this construction of a new state space is the following. If one has an orbit, coming from an autonomous set of equations, seen projected onto a single axis $\Omega(\cdot)$ or $s(n)$ on which the measurements happen to be made, then the orbit may have overlaps with itself in the variables $s(n)$ by virtue of the projection, not from the dynamics (i.e., two points quite far apart in original space may be projected near each other along the axis of scalar observations, Figure 3.1). As a consequence of the uniqueness theorems about the solutions of autonomous equations, there are no intersections of the orbit with itself in the true set of state variables. By providing enough independent coordinates $m$ for a multidimensional space made up of the observations, we can undo the overlaps resulting from the projection and recover orbits which are not ambiguous.

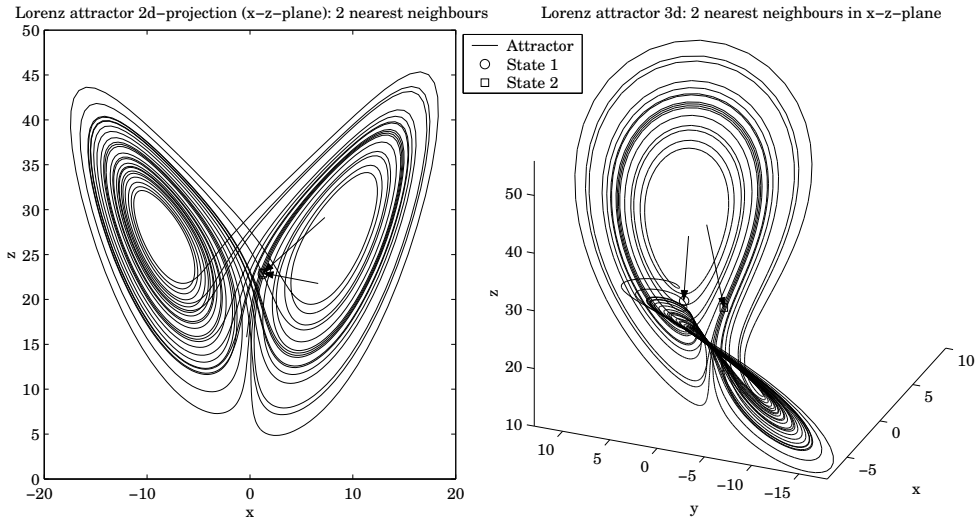


Figure 3.1: *Projection of the Lorenz attractor (equations 4.4-4.6) onto the x-z-plane (left). Two nearest neighbours in the projection are far apart in real state space (right). The projection introduces overlaps that do not exist in the true state space* $\mathcal{P} \subseteq \mathbb{R}^3$.

### Practical Choice of Delay $\tau$ and Embedding Dimension $m$

**Autocorrelation and Mutual Information: Choosing the Time Delay $\tau$.** Takens' theorem requires the time series to be noise free and that an infinite amount of data is available. Under these idealised conditions, it allows for any choice of time delay $\tau$ to use in constructing the $m$-dimensional time delay vectors $\boldsymbol{y}(n)$. In practice, however, a reasonable choice of the delay gains importance through the fact that we always have to deal with a finite amount of possibly noisy data with finite precision. The choice is guided by the following considerations:

1. The time lag has to be a multiple of the sampling time $t_s$.
2. If the time delay $\tau$ is too short, the coordinates we wish to use in our reconstructed vector will not be independent enough. Not enough time will have evolved for the system to have explored enough of its state space to produce new information about it.
3. Since chaotic systems are intrinsically unstable, if $\tau$ is too large, any connection between the measurements $s(n)$ and $s(n+T)$ is lost. The vectors $\boldsymbol{y}(n)$ will thus be (seemingly) randomly distributed in the embedding space $\mathcal{R}$.

A common, but ad hoc, choice for $\tau$ is the time step $T$ at which the autocorrelation function,

$$C(T) = \sum_{n=1}^{N} (s(n) - \bar{s}_N)(s(n+T) - \bar{s}_N)$$

$$\bar{s}_N = \frac{1}{N} \sum_{n=1}^{N} s(n),$$

has its first zero. Then the coordinates are linearly uncorrelated. The choice of the first zero of the autocorrelation function is the optimum linear choice from the point of view of predictability of $s(n+T)$ from knowledge of $s(n)$, in a least squares sense. What such a linear choice has to do with the nonlinear process relating $s(n)$ and $s(n+T)$ is, however, not clear at all [1].

A more sophisticated prescription for $\tau$ is given by the *average mutual information*. Unlike the autocorrelation function, it also takes into account nonlinear correlations. The mutual information between a measurement $a_i$ drawn from a set $A = \{a_i\}$ and a measurement $b_j$ drawn from a set $B = \{b_j\}$ is the amount learned by the measurement of $a_i$ about the measurement of $b_j$. In bits this is $\log_2 \frac{P_{AB}(a_i,b_j)}{P_A(a_i)P_B(b_j)}$, where $P_{AB}(a,b)$ is the joint probability density for measurements from $A$ and $B$, and $P_A(a)$ and $P_B(b)$ are the individual probability densities. The average mutual information between measurements from $A$ and measurements from $B$ is

$$I_{AB} = \sum_{i,j} P_{AB}(a_i, b_j) \log_2 \frac{P_{AB}(a_i, b_j)}{P_A(a_i)P_B(b_j)}.$$

In a deterministic system, we evaluate these "probabilities" by constructing a histogram of the variations of $a_i$ or $b_j$ seen in their measurements. Taking as the set of measurements $A$ the values of the observable $s(n)$, and for the measurements $B$ the values of $s(n+T)$, the amount (in bits) learned by measurements of $s(n)$ through measurements of $s(n+T)$ is

$$I(T) = \sum_{n} P(s(n), s(n+T)) \log_2 \frac{P(s(n), s(n+T))}{P(s(n))P(s(n+T))}.$$

The choice suggested is to take $\tau$ as the first minimum of the average mutual information $I(T)$. There exist good arguments that if the average mutual information exhibits a marked minimum at a certain value of $T$, then this is a good candidate for a reasonable time delay $\tau$. Still, there is no guarantee that the mutual information has a clear cut minimum, given the time resolution $t_s$ [1] [26] [32].

**False Nearest Neighbours: Choosing the Embedding Dimension $m$.** Takens' embedding theorem gives a sufficient condition on the embedding dimension $m$, if one knows the fractal dimension $d_A$ of the dynamical system underlying the time series under consideration. The condition is not necessary, and in practice the fractal dimension will often be unknown. Therefore, we wish to have a method to determine directly from the data the integer dimension at which we have the necessary number of coordinates to unfold the observed orbits from overlaps arising from projection of the attractor to a lower dimensional space. One such method is the method of *false nearest neighbours*.

Suppose the minimal embedding dimension for a given time series is $m_0$. This means that in an $m_0$-dimensional time delay space $\mathcal{R}$ the reconstructed attractor is a one-to-one image of the attractor in the original space $\mathcal{P}$. In particular, the topological properties are preserved. Thus, the neighbours of a given point in $\mathcal{P}$ are mapped onto neighbours in the reconstructed space $\mathcal{R}$. Suppose now we embed in an $m'$-dimensional space with $m' < m_0$. For this embedding the topological structure is no longer preserved. Therefore, points are projected into neighbourhoods of other points to which they would not belong

in higher dimensions $m \geq m_0$. These points are called *false neighbours*. Thus, what we have to find is the lowest dimension at which no false neighbours occur.

In practice, we start by embedding the time series with low dimension $m_1$. For each vector

$$\boldsymbol{y}_{m_1}(n) = [s(n), s(n-\tau), \cdots, s(n-(m_1-1)\tau)]$$

we find the $u$ nearest neighbours and check whether in the next higher embedding of dimension $m_1+1$ they are false neighbours or not[4]. The procedure is iterated up to an embedding dimension $m^*$ at which the percentage of false neighbours drops to zero. At this dimension we assume to have the necessary number of coordinates for unfolding the observed orbits.

### 3.2.2 Function Approximation Problem

In the previous subsection we have described a method for constructing an embedding space $\mathcal{R}$ for the scalar observations $s(n)$ in which the orbits of the dynamical system are unfolded, and we have shortly talked about practical considerations concerning this technique. Once an appropriate embedding for the observations is found, solving the function approximation problem means finding a prediction function (3.6)

$$\hat{P}_{\tau_p}(\boldsymbol{y}(n)) : \mathbb{R}^m \mapsto \mathbb{R} \tag{3.7}$$

that efficiently approximates (3.5),

$$P_{\tau_p}(\boldsymbol{y}(n)) = \Omega(\boldsymbol{f}_{\tau_p}(\boldsymbol{x}(n))), \tag{3.8}$$

using the available data. For chaotic systems, $P_{\tau_p}$, and therefore $\hat{P}_{\tau_p}$, is necessarily non-linear[5].

Assume we are given a time series $S = \{s(1), \cdots, s(N)\}$ and have found an embedding space of dimension $m$, using a delay $\tau$. The number of embedding vectors $\boldsymbol{y}(n)$ is therefore

$$M = N - (m-1)\tau,$$

and the number of samples for the map $s(n+\tau_p) = P_{\tau_p}(\boldsymbol{y}(n))$ is

$$N_E = N - (m-1)\tau - \tau_p.$$

These samples $\{(\boldsymbol{y}(n), s(n+\tau_p))\}$ can be used to estimate a prediction function $\hat{s}(n+\tau_p) = \hat{P}_{\tau_p}(\boldsymbol{y}(n))$.

We will now give a short non-exhaustive overview over possible methods for estimating the prediction function (3.7), and pointers to literature in which these techniques have been successfully applied for predicting chaotic time series.

[4]The nearest neighbours of a vector $\boldsymbol{y}(n)$ are the set of vectors $S_{\boldsymbol{y}}^{\mathrm{NN}} \subset S_{\boldsymbol{y}}$, indexed by $n_{\mathrm{NN}}$, with $S_{\boldsymbol{y}} = \{\boldsymbol{y}(1), \cdots, \boldsymbol{y}(M)\}$ being the set of all available embedding vectors $\boldsymbol{y}$, for which $\|\boldsymbol{y}(n) - \boldsymbol{y}(\tilde{n})\| \leq \|\boldsymbol{y}(n) - \boldsymbol{y}(m)\|$, $\tilde{n} \in n_{\mathrm{NN}}$, $m \in \{1, \cdots, M\} \setminus n_{\mathrm{NN}}$

[5]A linear prediction function (3.7) can approximate (3.8) only locally.

### Global Methods

One obvious possibility is to find one approximation $\hat{P}_{\tau_p}$, estimated from all $N_E$ available samples, that is valid in the entire reconstructed state space. Methods fitting one model to the complete state space are called *global methods*. They are necessarily nonlinear.

**Global Polynomials.** A possible approximation for (3.8) is a polynomial of degree $d$ in the $m$ delay coordinate variables,

$$\hat{s}(n+\tau_p) = \hat{P}_{\tau_p}(\boldsymbol{y}(n)) = \sum_i w_i \phi_i(\boldsymbol{y}(n)),$$

where $w_i$ are the parameters. The basis functions $\phi_i$ are powers and cross-products of the components in $\boldsymbol{y}(n)$. Since the parameters $w_i$ enter linearly, they can be fit to the data using standard least squares techniques. A disadvantage of polynomials is that the number of independent parameters equals $\binom{m+d}{m}$, which for large degree $d$ becomes computationally intractable and increases the risk of overfitting.

**Neural Networks (NN).** Another class of global methods are neural networks. They typically employ sigmoid shaped basis functions, e.g. $\phi(x) = \tanh(x)$ or $\phi(x) = \frac{1}{1+\exp^{-x}}$, and have an elaborate network structure of input layer, output layer, and hidden layers, where the outputs of one layer are used as weighted inputs for the nodes of the next layer. In contrast to global polynomials, the weights in neural networks do not enter linearly, so that iterative parameter estimation methods are required, among which the backpropagation algorithm is probably the most widely used one.

Neural networks may approximate any smooth function to any degree of accuracy, given enough sigmoid functions with accompanying weights. Typical disadvantages of this method are the long parameter estimation time, potential local minima, and the risk of overfitting, which can be reduced by regularisation techniques. Neural networks and the slightly different functional networks [7] have been considered for chaotic time series prediction in numerous variants, with generally high performance. The first to apply neural networks to prediction were Lapedes and Farber in [34]. Among other work, we mention [2][43][72], using standard neural networks, and [3][21][24] considering recurrent neural networks for this problem. Extensions to the classical neural networks for prediction are e.g. the usage of violation-guided learning algorithms [71], minimum description length neural networks [57][58], and neural networks employing Markov probabilities [14][73].

**Radial Basis Functions (RBF).** Applying $s$ basis functions, RBF approximants take the form

$$\hat{s}(n+\tau_p) = \hat{P}_{\tau_p}(\boldsymbol{y}(n)) = \sum_{i=1}^{s} w_i \phi_i(\|\boldsymbol{y}(n) - \boldsymbol{\zeta}^i\|),$$

where $\phi_i(r)$ are the radial symmetric basis functions centered at $\boldsymbol{\zeta}^i$. Different types of RBF functions $\phi_i$ lead to universal approximants. RBF are global methods with good location properties, and were e.g. used as RBF networks in [44] and [48]. Depending on the type of radial basis functions used, the parameters $w_i$ may not enter linearly, and time consuming parameter estimation techniques may become necessary.

**Support Vector Machines.** The function approximation problem (3.7-3.8) can be interpreted as an $m$-dimensional regression problem, which will be made clear in the next subsection. This can be efficiently solved within the framework of SVMs, as discussed in Section 2.3.

SVMs for regression estimation have been considered for chaotic time series prediction in various works in literature. Standard SVMs for chaotic time series prediction has e.g. been used in [42][65]. In [64] a modified SVM with "forgetting window" has been applied to financial time series prediction. Ralaivola et al. [47] have used SVMs for determining the preimage from a kernel space, in which linear Kalman filters performed chaotic time series prediction. The kernel recursive least squares algorithm has been employed and compared with standard SVM technique in [17].

### Local Methods

One of the main disadvantages of global methods is that changing a training sample may change the estimate everywhere in the embedding space $\mathcal{R}$. Local approximants overcome this drawback by using only a limited number of neighbouring samples for estimating a local prediction function for each single prediction, therefore working only in a subspace of the embedding space. Moreover, approximating the prediction function (3.8) locally may be easier than finding a predictor for the entire state space (Footnote 5). We can distinguish three approaches towards local prediction.

1. The simplest way to predict is to identify the nearest neighbour in embedding space, and to use it as a predictor.

2. An improvement is to find $u$ nearest neighbours and to use the average of their single step evolution as a predictor, which has been employed in e.g. [40].

3. The most successful local methods fit a function to approximate the single step evolution from $u$ nearest neighbours. The simplest local fit is a local linear model and was first reported in [18]. In [50] local linear models have been successfully used for the Santa Fe benchmark on time series prediction.

Note that all nonlinear function approximation methods described as global methods can be made local by estimating the prediction function locally from nearest neighbours. Local $\nu$-SVMs for predicting chaotic time series benchmarks are used in [19]. In [4], local nonlinear regression is employed for this task. A probabilistic framework to determine a neighbourhood called "relevance vectors" is described in [46] and uses kernel regression for prediction. Local functional networks have been considered in [39].

Another possibility for introducing locality is to partition the embedding space into a number of subspaces, within which "experts" are trained to approximate the (local) prediction functions. For performing prediction, one first determines which expert the state vector under consideration belongs to, and then uses this expert for prediction. In [6], this method is described for SVMs as experts, and neural networks to determine the subspaces of validity for the experts.

**Direct vs. Iterated Prediction**

Assume we are given a time series $\{s(n)\}_{n=1}^{N}$ and asked to provide a continuation. Maps (3.7) approximating (3.8) are one-step predictors. We can apply our prediction function $\hat{P}_{\tau_p}$ to predict a step $\tau_p$ ahead and get an estimate $\hat{s}(N+\tau_p)$. If it is desired to predict more, e.g. $r$ steps, ahead in spite of the increasing uncertainty, there are two obvious possibilities. First, we can repeat the one-step prediction $r$ times (*iterated prediction*), using previously estimated predictions as inputs to the single step predictor (3.7). Alternatively, we may estimate the first $r$ predictions directly, which means finding $r$ different prediction functions $\hat{P}_1, \hat{P}_2, \cdots, \hat{P}_r$ (*direct prediction*). The reliability of direct prediction is suspect because it is forced to predict farther ahead. When the prediction horizon $H$ increases, the function $P_H(\cdot) = \Omega(\boldsymbol{f}_H(\cdot))$ gets very complex and hard to approximate [35]. On the other hand, iterated prediction has to use the estimates $\hat{s}(N+1), \cdots, \hat{s}(N+r-\tau_p)$, which are possibly corrupted. Farmer [18] argues that iterated prediction is superior, although under ideal conditions that may not be realised in practice.

We will only consider iterated prediction with $\tau_p = 1$ in all experiments reported below.

## 3.3 Support Vector Machines for Chaotic Time Series Prediction

In the previous two sections, we have developed the framework necessary for predicting chaotic time series. We will now show how support vector regression machines can be applied to this problem.

### 3.3.1 Time Series Prediction as a Regression Problem

Suppose we are given a time series $\{s(n)\}_{n=1}^{N}$ of length $N$, and suppose it is embedded in an embedding space of dimension $m$, with a time delay $\tau$. Thus, we are given $M = N - (m-1)\tau$ embedding vectors $\{\boldsymbol{y}(n)\}$ of the form $\boldsymbol{y}(n) = [s(n), s(n-\tau), \cdots, s(n-(m-1)\tau)]$. Moreover, we have access to $N_E = N - (m-1)\tau - \tau_p$ samples $(\boldsymbol{y}(n), s(n+\tau_p))$ for the input-output mapping of a single step predictor (3.8),

$$\begin{aligned} P_{\tau_p}(\cdot) &: \mathbb{R}^m \mapsto \mathbb{R} \\ s(n+\tau_p) &= P_{\tau_p}(\boldsymbol{y}(n)) = P_{\tau_p}(s(n), s(n-\tau), \cdots, s(n-(m-1)\tau)), \end{aligned}$$

predicting $\tau_p$ steps ahead. The function approximation problem (3.3-3.6) can therefore be viewed as an $m$-dimensional regression estimation problem $[s(n), s(n-\tau), \cdots, s(n-(m-1)\tau)] \mapsto s(n+\tau_p)$ (Figure 3.2) with a training set of available input-output pairings

$$S = \{(\boldsymbol{y}(n), s(n+\tau_p))\}_{n=(m-1)\tau+1}^{N-\tau_p}. \tag{3.9}$$

The support vector regressor for this problem is of form (2.25),

$$\hat{s}(k+\tau_p) = \hat{P}_{\tau_p}(\boldsymbol{y}(k)) = h(\boldsymbol{y}(k)) = \sum_{n=(m-1)\tau+1}^{N-\tau_p} \beta_n^* K(\boldsymbol{y}(n), \boldsymbol{y}(k)) + b^*, \quad k > N-\tau_p, \tag{3.10}$$

where $\beta_i^*$ and $b^*$ are the solutions to the optimisation problems in Proposition 15 and 16, respectively, and $K(\cdot,\cdot)$ is a kernel[6].

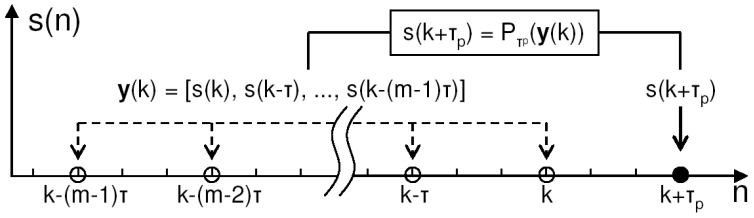


Figure 3.2: *Time series prediction from embedding vectors $\boldsymbol{y}(n)$ can be interpreted as a regression estimation problem $[s(n), s(n-\tau), \cdots, s(n-(m-1)\tau)] \mapsto s(n+\tau_p)$.*

**Global Support Vector Prediction.** The global support vector predictor is found by solving the optimisation problems in Proposition 15 and 16 for the entire available training data (3.9). Once the $\beta_i^*$ and $b^*$ are found, the prediction $\hat{s}(k+\tau_p)$ is obtained by evaluating the hypothesis (3.10) at the desired position $\boldsymbol{y}(k)$. The prediction $\hat{s}(k)$ is

$$\hat{s}(k) = h(\boldsymbol{y}(k-\tau_p)) = \hat{P}_{\tau_p}(\boldsymbol{y}(k-\tau_p)) = \tag{3.11}$$

$$= \sum_{n=(m-1)\tau+1}^{N-\tau_p} \beta_n^* K(\boldsymbol{y}(n), \boldsymbol{y}(k-\tau_p)) + b^* = \tag{3.12}$$

$$= \sum_{n=(m-1)\tau+1}^{N-\tau_p} \beta_n^* K\left(\begin{bmatrix} s(n) \\ s(n-\tau) \\ \vdots \\ s(n-(m-1)\tau) \end{bmatrix}, \begin{bmatrix} s(k-\tau_p) \\ s(k-\tau_p-\tau) \\ \vdots \\ s(k-\tau_p-(m-1)\tau) \end{bmatrix}\right) + b^*. \tag{3.13}$$

The first $\tau_p$ predictions $\hat{s}(k)$, $k = N+1, \cdots, N+\tau_p$, are obtained by predicting from embedding vectors $\boldsymbol{y}(k-\tau_p)$ contained in the training set (3.9). For obtaining predictions further ahead ($\hat{s}(k)$ for $k > N+\tau_p$), the predictor (3.11-3.13) is iterated on embedding vectors $\hat{\boldsymbol{y}}(k-\tau_p) = [\overset{(\wedge)}{s}(k-\tau_p), \overset{(\wedge)}{s}(k-\tau_p-\tau), \cdots, \overset{(\wedge)}{s}(k-\tau_p-(m-1)\tau)]$, whose coordinates are themselves predictions $\hat{s}(k)$ for $k > N$. The procedure is illustrated in the left column of Figure 3.3.

**Local Support Vector Prediction.** For each single prediction step, a new local predictor, $\hat{s}(k+\tau_p) = h_k(\boldsymbol{y}(k))$ is found by solving the optimisation problems of Proposition 15 and 16 for the local training set

$$S_{NN}(k) = \{(\boldsymbol{y}(n), s(n+\tau_p))\}_{n \in n_{\mathrm{NN}}(k)}.$$

[6]We call hypotheses of form (3.9) SVR predictors, support vector predictors or simply predictors when the context makes it clear that we are talking about SVR predictors.

It consists of the $u$ nearest neighbours of $\boldsymbol{y}(k)$ in the training set (3.9) indexed by $n_{\mathrm{NN}}(k)$. When the $\beta_n^*$ and $b^*$ for the local support vector predictor are found, the prediction is

$$\hat{s}(k) = h_k(\boldsymbol{y}(k-\tau_p)) = \tag{3.14}$$

$$= \sum_{n \in n_{\mathrm{NN}}(k-\tau_p)} \beta_n^* K \left( \begin{bmatrix} s(n) \\ s(n-\tau) \\ \vdots \\ s(n-(m-1)\tau) \end{bmatrix}, \begin{bmatrix} s(k-\tau_p) \\ s(k-\tau_p-\tau) \\ \vdots \\ s(k-\tau_p-(m-1)\tau) \end{bmatrix} \right) + b^*. \tag{3.15}$$

As discussed for the global support vector predictor, predictions of observations that are further ahead in time ($\hat{s}(k)$ for $k > N + \tau_p$) are obtained by iterating the predictor (3.14-3.15) on embedding vectors whose coordinates are themselves predictions $\hat{s}(k)$ for $k > N$. The procedure is illustrated in the right column of Figure 3.3.

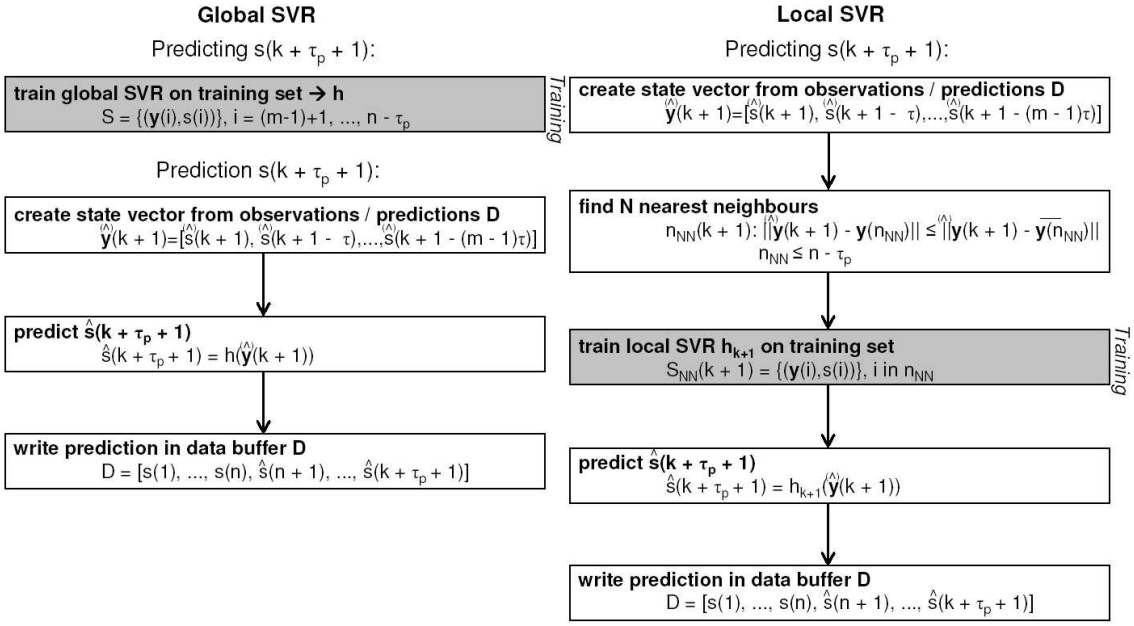


Figure 3.3: *Flow chart for time series prediction with SVR: global model (left) and local model (right). In the global case, one single predictor is trained on the entire training set, whereas in the local case, a new local predictor is trained on a local training set of nearest neighbours for each prediction iteration.*

# Chapter 4

# Results

## 4.1 Experiments

### 4.1.1 Simulating Chaotic Systems and Evaluating Prediction Performance

**Numerical Integration of Chaotic Systems.** If we want to perform prediction on a synthetic, continuous-time dynamical system, we first have to calculate trajectories by numerical simulation. The integration of chaotic systems poses a special problem, since its sensitive dependence on initial conditions implies that any arbitrarily small error may grow exponentially and affect the behaviour of the simulated system. Therefore, no integration routine can estimate with any accuracy the state of a chaotic system after a long period of integration. Despite the fact that every integration routine introduces an error, if the local integration error is small, the output points of the integration routine may not lie on one "true" trajectory, but still approach the attractor, as a strange attractor is attracting. Nevertheless, simulation of chaotic systems requires careful interpretation [45]. Figure 4.1 illustrates this problematics. It shows the $x$-coordinate of the Lorenz attractor, obtained by a fourth order Runge-Kutta numerical integration of the Lorenz equations (4.4-4.6) on two different machines.

**Prediction Performance Measures.** It is not immediately evident how to evaluate the performance of a predictor on a dynamical system exhibiting chaotic behaviour.

One possibility of performance evaluation is in terms of whether the predictor has captured the dynamics and preserves the invariant measures of the system, such as fractal dimension and global Lyapunov exponents. Calculating the invariant measures from the prediction is not trivial, requires large amount of data and computation time [1]. Invariant measures are therefore not considered by many authors, or assessed visually through phase plots.

Another commonly used choice of performance measures are the single step prediction error and iterated prediction error, expressing how close the prediction is to the true series. They have the advantage of being easy to assess and to compare. What is more, in many applications it is of great interest how closely the prediction follows the true observation[1].

[1]For instance, think of a financial time series, where a deviation of the prediction from the true series

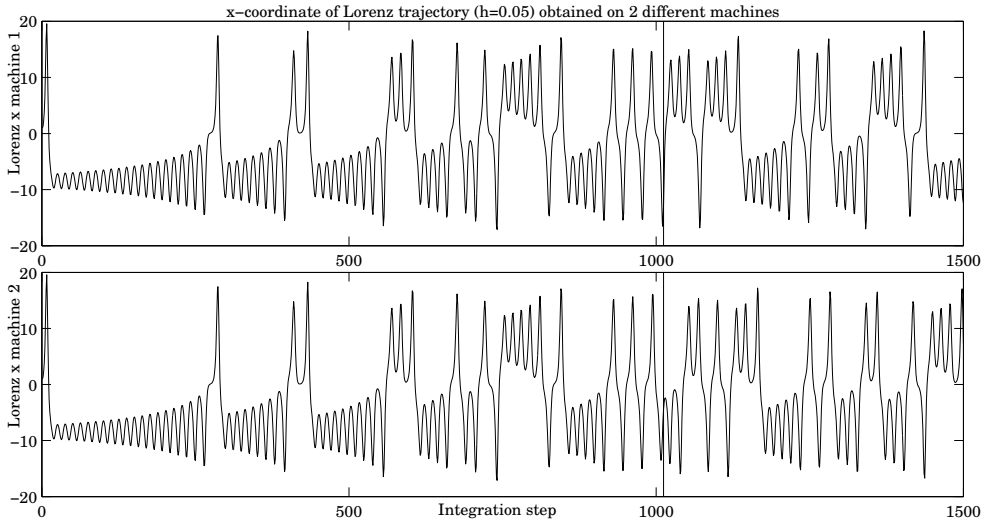


Figure 4.1: *Numerical integration of Lorenz equations (4.4-4.6) obtained on two different machines. The trajectories diverge after approximately 1000 steps. Both trajectories stay confined to the region determined by the strange attractor.*

A widely used prediction error measure is the *normalized mean squared error* (NMSE). Given observations $\{s(n)\}_{n=1}^{N}$ to be predicted, the normalized mean squared error is defined as

$$\text{NMSE} = \frac{1}{\hat{\sigma}_N^2 N} \sum_{n=1}^{N} (s(n) - \hat{s}(n))^2,$$

where $s(n)$ and $\hat{s}(n)$ are the observations and the predictions, respectively, and

$$\hat{\sigma}_N^2 = \frac{1}{N} \sum_{n=1}^{N} (s(n) - \bar{s}_N)^2, \quad \bar{s}_N = \frac{1}{N} \sum_{n=1}^{N} s(n)$$

are the sample variance and the sample mean of the observations in the prediction interval considered. A value of NMSE = 1 corresponds to simply predicting the average. The normalized mean squared error has the advantage over other error measures, e.g. the mean squared error, that it is insensitive to rescaling of the data [74].

Unfortunately, there is quite a large number of literature in which custom error measures, e.g. a "fit" expressed in percent [16], are used, which renders comparisons very difficult.

In our experiments, we use the normalized mean squared error for single step prediction and iterated prediction as a performance measure. If appropriate, we consider plots for qualitatively evaluating how well a predictor captures the dynamics of the system.

### 4.1.2 The Support Vector Predictors

For all experiments reported below, we use global and local support vector predictors (3.11-3.13) and (3.14-3.15). We employ standard polynomial and Gaussian kernels (2.51)

essentially incurs loss of money due to the unexpected behaviour of the quantity, e.g. a currency exchange rate

and (2.52), and a linear $\epsilon$-insensitive loss function. All results are obtained with SVM-light [29] (http://svmlight.joachims.org), a fast implementation of the support vector algorithms for classification and regression written in C, through a MATLAB® interface.

After properly embedding a specific time series, prediction with SVMs consists of two phases: a parameter evaluation phase, and a prediction phase.

**Parameter Evaluation Phase.** In the parameter evaluation phase, we perform a grid search with simple hold out validation (Section 2.6) to select appropriate hyperparameters for the SVM[2]. For a time series of $N_{\text{train}}$ observations, the SVM is trained on the first $N_{\text{gridtraining}} = N_{\text{train}} - (m-1)\tau - \tau_p - N_{\text{valid}}$ training samples $(\boldsymbol{y}(n), s(n+\tau_p))$. Then, a prediction is performed on the validation set consisting of the remaining $N_{\text{valid}}$ samples $s(n)$, and the performance of the hyperparameter setting is assessed by the single step prediction error on this validation set. The hyperparameter setting with the lowest single step prediction error is selected and used for the prediction phase.

**Prediction Phase.** In the prediction phase, the final prediction on the time series is obtained. The SVM is trained on all $N_{\text{train}}$ available observations, using the hyperparameter setting selected in the parameter evaluation phase. Then, the prediction is performed on the test set(s) of length $N_{\text{test}}$. If applicable, we measure the prediction performance as the average of the prediction error on $N_{\text{run}}$ subsequent test sets[3].

### 4.1.3 Data Sets and Experimental Setup

The time series we consider are chosen in order to allow for comparison with previously obtained results described in literature.

#### The Hénon Map

The Hénon map [27] is a two-dimensional quadratic map given by the equations

$$x(n+1) = 1 - ax(n)^2 + by(n) \tag{4.1}$$
$$y(n+1) = x(n). \tag{4.2}$$

It is one of the simplest 2-dimensional invertible maps and has been studied in detail for the parameter setting $a = 1.4$ and $b = 0.3$, where numerical evidence of chaotic behaviour was found.

The fractal dimension of the attractor is $d_A \approx 1.26$ [49]. Therefore, the embedding dimension according to Takens' theorem is $m = 4$. The Hénon attractor can, however, be successfully reconstructed with an embedding dimension as low as $m = 2$, if the coordinates of the time delay vectors are chosen to be $\boldsymbol{y}(n) = [x(n), x(n-1)]$, and $\tau_p$ is chosen to be 1 [32]. This does not come as a surprise, since the observable $x(n+1)$ is completely determined by the knowledge of $x(n)$ and $x(n-1)$.

[2]Apart from the hyperparameters for the SVM, the number of nearest neighbours $u$ for the local predictors is chosen through the grid search. For data set A, we consider as well the embedding dimension $m$ as a parameter to be assessed by the grid search.

[3]For Santa Fe data set A, $N_{\text{run}} = 1$ (see Subsection 4.1.3).

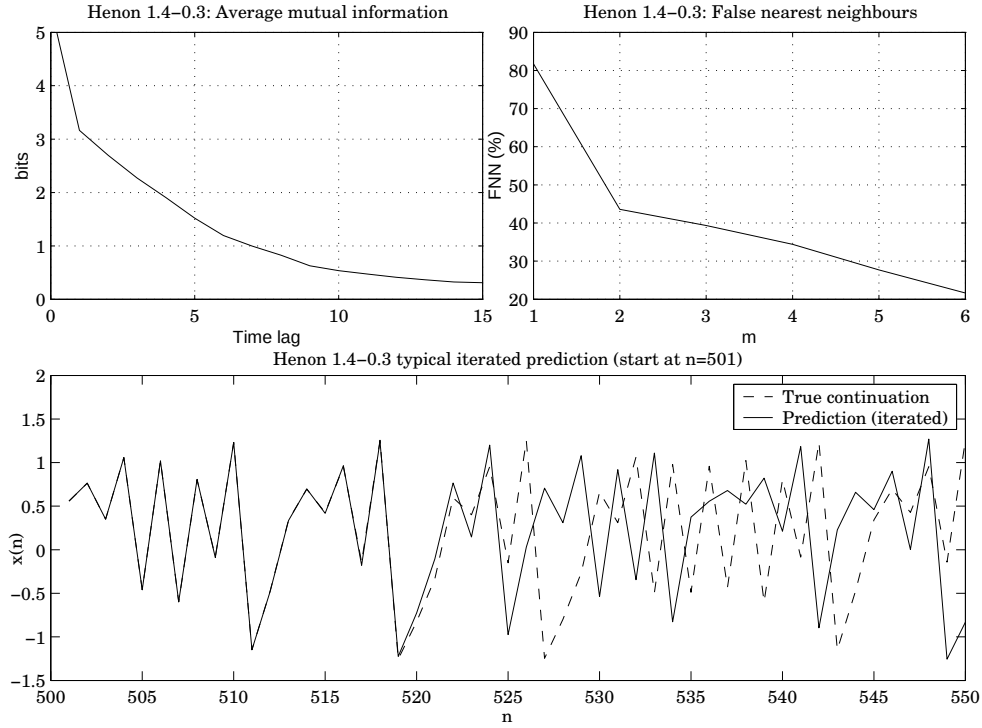


Figure 4.2: *Average mutual information and false nearest neighbours analysis of Hénon time series (top). The average mutual information does not possess a clear cut minimum, and the false nearest neighbours analysis suggests an embedding dimension $m > 6$. A part of the Hénon time series and a typical iterated prediction on it (bottom), obtained with a global Gaussian SVR. The prediction (solid line) loses the time series (dashed line) after approximately* 18 *iterations.*

At a first look, the analysis of the average mutual information does not confirm this specific choice for $\tau$, since it does not have a clear cut minimum at all (Figure 4.2 top left). It is, however, observed that in such a case the delay should in fact be set to $\tau = 1$.

The false nearest neighbour analysis would suggest values $m > 6$ (Figure 4.2 top right).

**Experimental Setup.** We follow the prescription of the average mutual information analysis and set $\tau = \tau_p = 1$. Since we know that the attractor can then be successfully reconstructed for $m \geq 2$, but the false nearest neighbours analysis suggests $m > 6$, we compare time delay embeddings with embedding dimensions $m \in \{2, 3, 4, 5, 6\}$.

The time series $s(n) = x(n)$ is obtained by iterating the map with initial conditions $x(0) = y(0) = 0$ and parameter values $a = 1.4$ and $b = 0.3$. The first 10000 samples are skipped to ensure that all transients have settled down[4]. We consider having a sequence $\{s(n)\}$ of $N_{\text{train}} = 500$ observations at our disposition.

For the grid search, this set is split into a set of 450 observations for training, giving

[4]Modelling the attractor from data with transients present is difficult and usually not considered in literature. We are only interested in the part of the time series for which we are already in the attractor. For most measured physical time series (e.g. electro-encephalogram (EEG)), it is ensured that the transients have already settled down.

$N_{\text{gridtrain}} = 450 - (m-1)\tau - \tau_p$ training samples $(\boldsymbol{y}(n), s(n+\tau_p))$, and a remaining set of $N_{\text{valid}} = 50$ observations for validation. The hyperparameter combination with the lowest single step prediction error on the validation set is chosen for the final prediction.

The prediction performance of the final predictors is assessed through $N_{\text{run}} = 50$ runs on subsequent test sets of length $N_{\text{test}} = 50$.

### The Mackey Glass Delay Differential Equation

The Mackey-Glass equation is a delay differential equation (dde) that has been proposed as a model of white blood cell production [38]. It is given by

$$\frac{ds}{dt} = \frac{as(t-T)}{1+s^c(t-T)} - bs(t). \tag{4.3}$$

The values of the constants $a$, $b$ and $c$ are usually taken as 0.2, 0.1 and 10, respectively. For different delay parameter $T$, the attractor possesses different fractal properties. If $T$ is chosen large, the fractal dimension $d_A$ of the attractor gets large as well, an attempts to predict the time series may not be useful since attractors with high fractal dimension usually have a large positive Lyapunov exponent, resulting in a very fast divergence of the trajectories (Table 4.1).

| $T$ | $d_A$ |
|---|---|
| 17 | 1.95 |
| 23 | 2.44 |
| 30 | 3.15 |
| 100 | 7.5 |

Table 4.1: *Fractal dimension $d_A$ of Mackey Glass delay differential equation for different time delays $T$ [32].*

**Experimental Setup.** To obtain the time series for our experiments, (4.3) is solved with the MATLAB® function *dde23*, with a delay $T = 17$ and a history vector with values $s(0) = \cdots = s(\Delta - 1) = 0.9$. The first 10000 samples are skipped to ensure that all transients have settled down, and the time series is downsampled[5] by 6. This ensures being consistent with earlier experiments reported in literature[6].

Figure 4.3 (top) shows that the false nearest neighbours and average mutual information analysis of the time series suggest to choose an embedding dimension of approximately $m > 6$, and a time delay of around $\tau = 2$. Preliminary experiments motivate us to use the values $m = 7$ and $\tau = \tau_p = 1$ instead.

We consider having a sequence $\{s(n)\}$ of $N_{\text{train}} = 500$ observations at our disposition. For the grid search, this set is split into a set of 450 observations for training, giving $N_{\text{gridtrain}} = 450 - (m-1)\tau - \tau_p$ training samples $(\boldsymbol{y}(n), s(n+\tau_p))$, and a remaining set of

[5] We shortly write $\text{MG}_{17}$ for this series.

[6] Note that predicting $n$ steps of a time series downsampled by 6, with $\tau = \tau_p = 1$, is (if we ignore differences in the training sets) equivalent to predicting $6n$ steps of the original series, with $\tau = \tau_p = 6$.

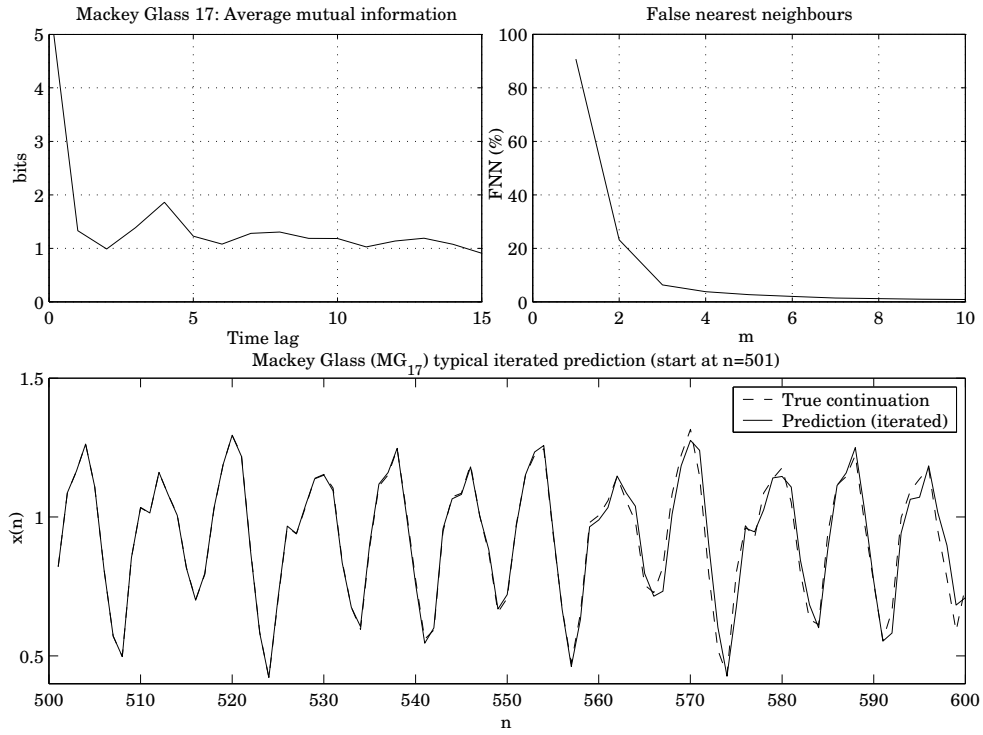


Figure 4.3: *Average mutual information and false nearest neighbours analysis of Mackey-Glass time series (top). The average mutual information has its first minimum at delay* $\tau = 2$*, and the percentage of false nearest neighbours drops to zero for embedding dimension* $m > 6$*. A part of the time series and a typical iterated prediction on it (bottom), obtained with a global Gaussian SVR. The prediction (solid line) is of very high accuracy up to iteration* 60 *and keeps on staying close to the true series (dashed line) when iterating further.*

$N_{\text{valid}} = 50$ observations for validation. The hyperparameter combination with the lowest single step prediction error on the validation set is chosen for the final prediction.

The prediction performance of the final predictors is assessed through $N_{\text{run}} = 50$ runs on subsequent test sets of length $N_{\text{test}} = 100$.

### The Lorenz Equations

The famous Lorenz equations [37] are the set of three simultaneous differential equations given by

$$\frac{dx}{dt} = -\sigma x + \sigma y \tag{4.4}$$

$$\frac{dy}{dt} = -xz + rx - y \tag{4.5}$$

$$\frac{dz}{dt} = xy - bz. \tag{4.6}$$

In the early 1960s, Lorenz accidentally discovered the chaotic behaviour of this system. It has fractal dimension $d_A = 2.06$ [22].

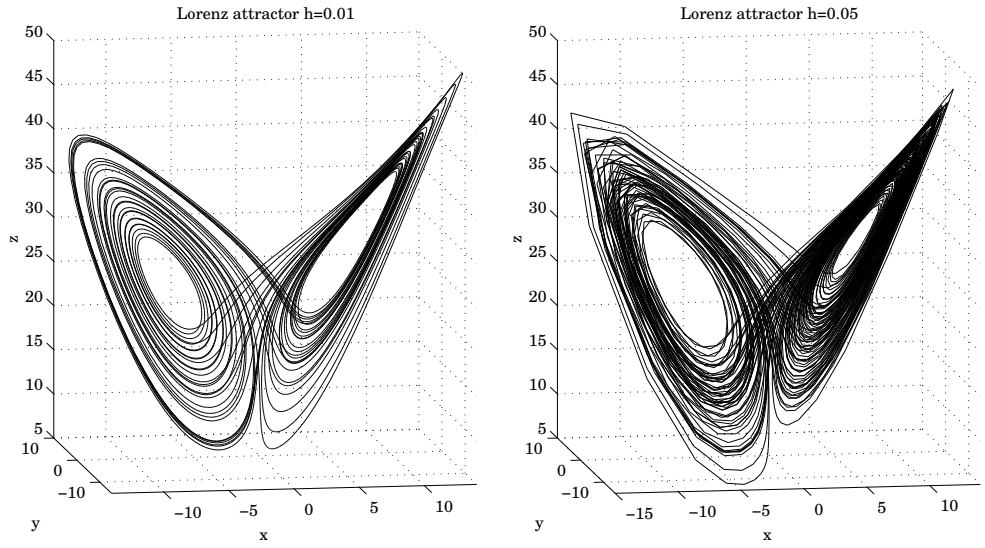


Figure 4.4: *The Lorenz attractor, obtained by numerical integration of the Lorenz equations (4.4-4.6) with integration time steps h=0.01 and h=0.05.*

**Experimental Setup.** We consider prediction on the time series $s(n) = x(n)$ given by the $x$-coordinate of the Lorenz system, with $\sigma = 10$, $r = 28$, $b = \frac{8}{3}$ and initial conditions $x(0) = y(0) = z(0) = 1$. The time series are obtained by a fourth order Runge-Kutta integration method with time steps $h = 0.01$ and $h = 0.05$. The first 10000 samples are skipped to ensure that all transients have settled down. The corresponding attractors are shown in Figure 4.4.

The false nearest neighbours and average mutual information analysis of the time series suggest to use an embedding dimension $m = 3$, and an time delay of approximately $\tau = 16$ and $\tau = 3$ for integration time step $h = 0.01$ and $h = 0.05$, respectively (Figure 4.5). Our numerical results have been obtained for $m = 3$, $\tau = 16$ and $\tau = 2$ for integration time step $h = 0.01$ and $h = 0.05$, respectively, and a prediction step size of $\tau_p = 1$.

The observed sequences $\{s(n)\}$ we use consist of $N_{\text{train}} = 1000$ samples each. For the grid search, these sets are split into sets of 900 observations for training, giving $N_{\text{gridtrain}} = 900 - (m-1)\tau - \tau_p$ training samples $(\boldsymbol{y}(n), s(n+\tau_p))$, and the remaining sets of $N_{\text{valid}} = 100$ observations for validation. The hyperparameter combinations with the lowest single step prediction error on the validation sets are chosen for the final predictions.

The prediction performance of the final predictors is assessed through $N_{\text{run}} = 50$ runs on subsequent test sets of length $N_{\text{test}} = 100$.

### Santa Fe Data Set A (Laser)

Data set A has been used as one of the benchmark time series in the Santa Fe competition [74]. Since then, it has been analysed in many publications. Data set A consists of samples of the fluctuations in a far-infrared laser, approximately described by three ordinary differential equations, and has been recorded in a laboratory experiment [25]. For the competition, the first 1000 samples of the series were given, and it was asked to provide a continuation of the subsequent 100 samples, starting at sample 1001 (Figure

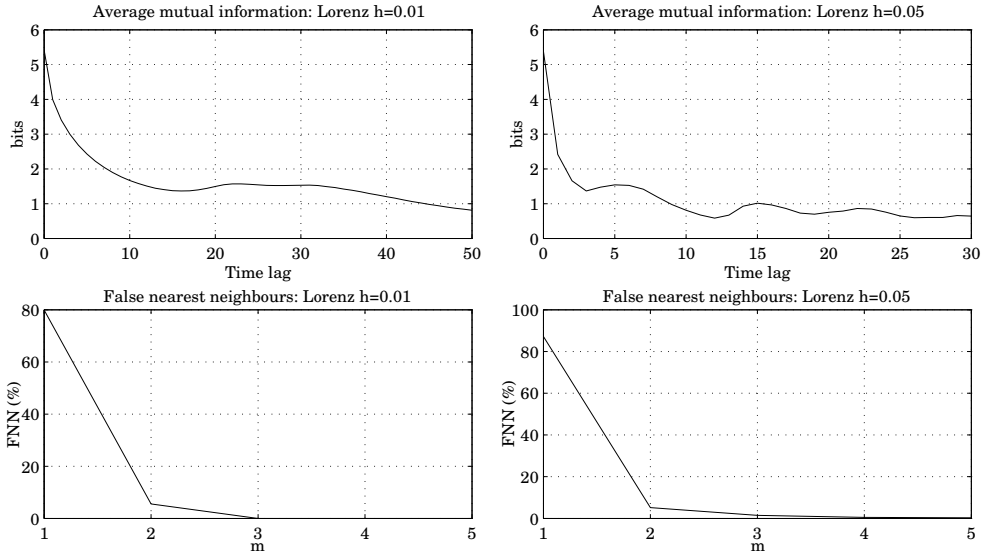


Figure 4.5: *Average mutual information and false nearest neighbours analysis of the x-coordinate of the Lorenz attractor for integration time steps h=0.01 (left) and h=0.05 (right). The average mutual information has its first minimum at delay* $\tau = 16$ *(*$h = 0.01$*) and* $\tau = 2$ *(*$h = 0.05$*). For both integration time steps, the percentage of false nearest neighbours drops to zero for embedding dimension* $m \geq 3$*.*

4.6, bottom). The problem is considered as hard, due to the presence of measurement noise, and since a collapse of the laser intensity has to be predicted in the test set and appears only once in the training set. In [31] it has been pointed out that this task is in fact a pattern matching problem, as the first 73 samples of the continuation are nearly identical to a sequence of 73 samples in the training set.

After the competition, a longer continuation has been provided for experiments, and four additional test sets have been established, starting at samples 2181, 3871, 4001 and 5181.

**Experimental Setup.** We follow the original setup of the Santa Fe competition and assume to have knowledge of only the first $N_{\text{train}} = 1000$ samples for choosing the embedding dimension and the time delay, and for training the predictor.

The analysis of the first 1000 samples of data set A suggests a delay of $\tau = 1$ or 2, and an embedding dimension of $m > 10$ (Figure 4.6). We set the delay to $\tau = 1$, and the prediction step to $\tau_p = 1$.

A good choice of the embedding dimension gains importance through the fact that the series is known to contain measurement noise. As the false nearest neighbours analysis just gives prescriptions, we decide to incorporate the embedding dimension $m$ as a parameter in the grid search. Embedding dimensions $m \in \{15, \cdots, 25\}$ are considered, and the model with the lowest single step prediction error on the validation set ($m^* = 18$) is chosen for the "competition" prediction that we compare with results reported in literature[7].

[7]The predictors obtained for the other embedding dimensions are not used for comparisons with results reported by other authors. They will be considered separately in the discussion on Figure 4.17.

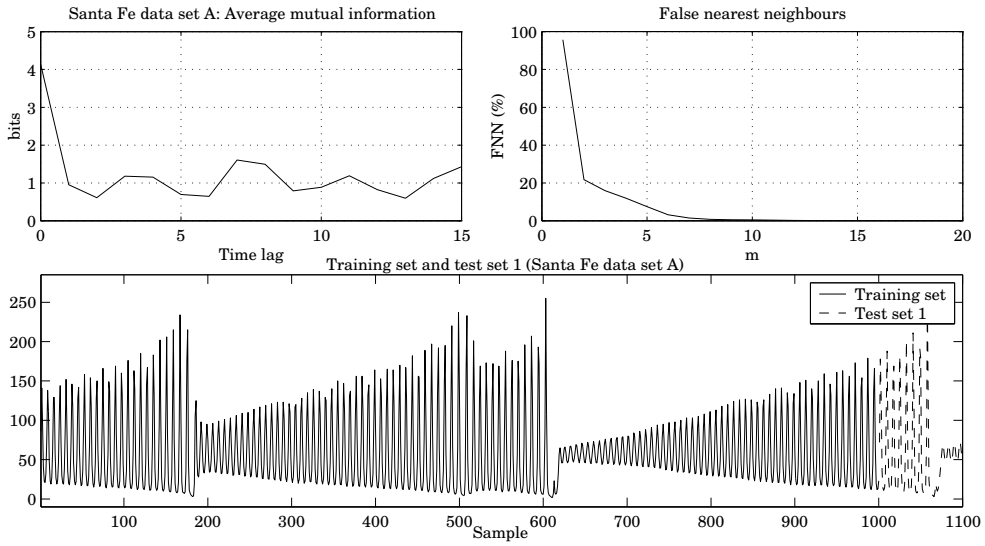


Figure 4.6: *Average mutual information and false nearest neighbours analysis of Santa Fe data set A (top). The analysis suggests using a time delay* $\tau = 1$ *or* 2*, and an embedding dimension* $m > 10$*. The training set and the first test set of data set A (bottom). The training set consists of the first* 1000 *samples of the time series (solid line), and the first test set of the subsequent* 100 *samples (dashed line)*

For the grid search, the set of $N_{\text{train}} = 1000$ samples given at the competition are split into a set of 900 observations for training, resulting in $N_{\text{gridtrain}} = 900 - (m-1)\tau - \tau_p$ training samples $(\boldsymbol{y}(n), s(n+\tau_p))$, and the remaining set of $N_{\text{valid}} = 100$ observations for validation. The best hyperparameter setting in terms of single step prediction error on the validation set is selected.

For the predictions, the SVMs are trained on all $N_{\text{train}} = 1000-(m-1)\tau-\tau_p$ available training samples. The performance is then evaluated on the five test sets of length $N_{\text{test}} = 100$ starting at samples 1001, 2181, 3871, 4001 and 5181, and reported separately[8].

## 4.2 Results

In this section, we summarise and discuss the single step and iterated prediction results on the data sets described in Subsection 4.1.3. We consider both global and local predictors, with Gaussian and polynomial kernels, for the experiment setups described above[9].

### 4.2.1 Prediction on the Hénon Time Series

The time series generated by the Hénon map and a typical iterated prediction on it, obtained with a global Gaussian predictor, are shown in Figure 4.2 (bottom). We observe

[8]We emphasize that the predictions on all test sets are obtained with a predictor that is trained only on the first 1000 samples of the series.

[9]We write shortly *global Gaussian*, *local Gaussian*, *global polynomial*, and *local polynomial* to distinguish between these predictors.

that the prediction is of high accuracy up to prediction step 18, and then loses the true trajectory. Nevertheless, the series produced by the iterated prediction appears to stay in the attractor even after it diverges from the true signal. In Table 4.2, results obtained with global and local Gaussian and polynomial predictors of dimension $m \in \{2, 3, 4, 5, 6\}$, averaged over 50 subsequent test sets, are summarised together with results reported by other authors. The performance of all predictors is compared in Figure 4.7.

| Method | m | $N_{\text{train}}$ | $N_{\text{run}} \cdot N_{\text{test}}$ | Single step | 5 step | 10 step | 20 step |
|---|---|---|---|---|---|---|---|
| Global G | 2 | 500 | $50 \cdot 50$ | $\mathbf{7.68 \cdot 10^{-7}}$ | $4.02 \cdot 10^{-5}$ | $2.56 \cdot 10^{-3}$ | $\mathbf{3.04 \cdot 10^{-1}}$ |
| | 3 | | | $1.09 \cdot 10^{-6}$ | $2.21 \cdot 10^{-5}$ | $1.80 \cdot 10^{-3}$ | $3.23 \cdot 10^{-1}$ |
| | 4 | | | $1.30 \cdot 10^{-6}$ | $\mathbf{2.00 \cdot 10^{-5}}$ | $\mathbf{1.62 \cdot 10^{-3}}$ | $3.17 \cdot 10^{-1}$ |
| | 5 | | | $8.30 \cdot 10^{-6}$ | $2.82 \cdot 10^{-5}$ | $2.18 \cdot 10^{-3}$ | $3.68 \cdot 10^{-1}$ |
| | 6 | | | $2.26 \cdot 10^{-5}$ | $3.72 \cdot 10^{-4}$ | $1.74 \cdot 10^{-2}$ | $3.76 \cdot 10^{-1}$ |
| Local G | 2 | 500 | $50 \cdot 50$ | $1.34 \cdot 10^{-6}$ | $9.05 \cdot 10^{-5}$ | $4.38 \cdot 10^{-3}$ | $4.65 \cdot 10^{-1}$ |
| | 3 | | | $\mathbf{1.15 \cdot 10^{-6}}$ | $4.73 \cdot 10^{-5}$ | $\mathbf{3.15 \cdot 10^{-3}}$ | $3.53 \cdot 10^{-1}$ |
| | 4 | | | $2.24 \cdot 10^{-6}$ | $\mathbf{4.37 \cdot 10^{-5}}$ | $4.62 \cdot 10^{-3}$ | $\mathbf{3.29 \cdot 10^{-1}}$ |
| | 5 | | | $1.17 \cdot 10^{-5}$ | $8.22 \cdot 10^{-5}$ | $4.63 \cdot 10^{-3}$ | $3.83 \cdot 10^{-1}$ |
| | 6 | | | $7.48 \cdot 10^{-5}$ | $2.43 \cdot 10^{-4}$ | $1.19 \cdot 10^{-2}$ | $4.26 \cdot 10^{-1}$ |
| Global p | 2 | 500 | $50 \cdot 50$ | $6.83 \cdot 10^{-7}$ | $5.19 \cdot 10^{-5}$ | $2.64 \cdot 10^{-3}$ | $5.40 \cdot 10^{-1}$ |
| | 3 | | | $7.95 \cdot 10^{-7}$ | $9.28 \cdot 10^{-5}$ | $4.65 \cdot 10^{-3}$ | $3.67 \cdot 10^{-1}$ |
| | 4 | | | $5.37 \cdot 10^{-7}$ | $3.06 \cdot 10^{-5}$ | $2.05 \cdot 10^{-3}$ | $4.13 \cdot 10^{-1}$ |
| | 5 | | | $7.53 \cdot 10^{-7}$ | $\mathbf{2.88 \cdot 10^{-5}}$ | $\mathbf{1.64 \cdot 10^{-3}}$ | $3.50 \cdot 10^{-1}$ |
| | 6 | | | $\mathbf{5.12 \cdot 10^{-7}}$ | $5.60 \cdot 10^{-5}$ | $3.91 \cdot 10^{-3}$ | $\mathbf{3.35 \cdot 10^{-1}}$ |
| Local p | 2 | 500 | $50 \cdot 50$ | $8.42 \cdot 10^{-7}$ | $4.22 \cdot 10^{-5}$ | $2.71 \cdot 10^{-3}$ | $3.62 \cdot 10^{-1}$ |
| | 3 | | | $\mathbf{7.69 \cdot 10^{-7}}$ | $5.09 \cdot 10^{-5}$ | $3.00 \cdot 10^{-3}$ | $4.35 \cdot 10^{-1}$ |
| | 4 | | | $9.60 \cdot 10^{-7}$ | $\mathbf{2.48 \cdot 10^{-5}}$ | $\mathbf{1.61 \cdot 10^{-3}}$ | $\mathbf{3.15 \cdot 10^{-1}}$ |
| | 5 | | | $8.67 \cdot 10^{-7}$ | $4.13 \cdot 10^{-5}$ | $2.51 \cdot 10^{-3}$ | $3.47 \cdot 10^{-1}$ |
| | 6 | | | $3.35 \cdot 10^{-5}$ | $1.67 \cdot 10^{-4}$ | $8.52 \cdot 10^{-3}$ | $3.32 \cdot 10^{-1}$ |
| RBF [44] | | 4000 | 1000 | $7.4 \cdot 10^{-5}$ | - | - | - |
| NN [71] | | 5000 | 5000 | $\mathbf{3.4 \cdot 10^{-5}}$ | - | - | - |

Table 4.2: *Comparison of prediction performance on Hénon map (a=1.4, b=0.3). All results are normalized mean squared errors. The test set and training set size are given in samples. The second part of the table gives the single step prediction error, and the third part the 5-, 10-, and 20-step iterated prediction error. Own results are averaged over 50 runs. "G" stands for Gaussian kernel, "p" for polynomial kernel. "RBF" stands for radial basis function network, "NN" for neural network. Best results are marked in bold for each part of the table.*

**Single Step Error and Iterated Prediction.** We note that the predictors that give the best single step prediction do not have lowest iterated prediction error. The predictors with worst single step prediction error, however, do as well have bad iterated prediction performance (Table 4.2). Thus, the single step prediction error is not clearly connected to the iterated prediction performance. This performance measure alone can therefore not give clear statements on how well a predictor captured the dynamics of the Hénon system.

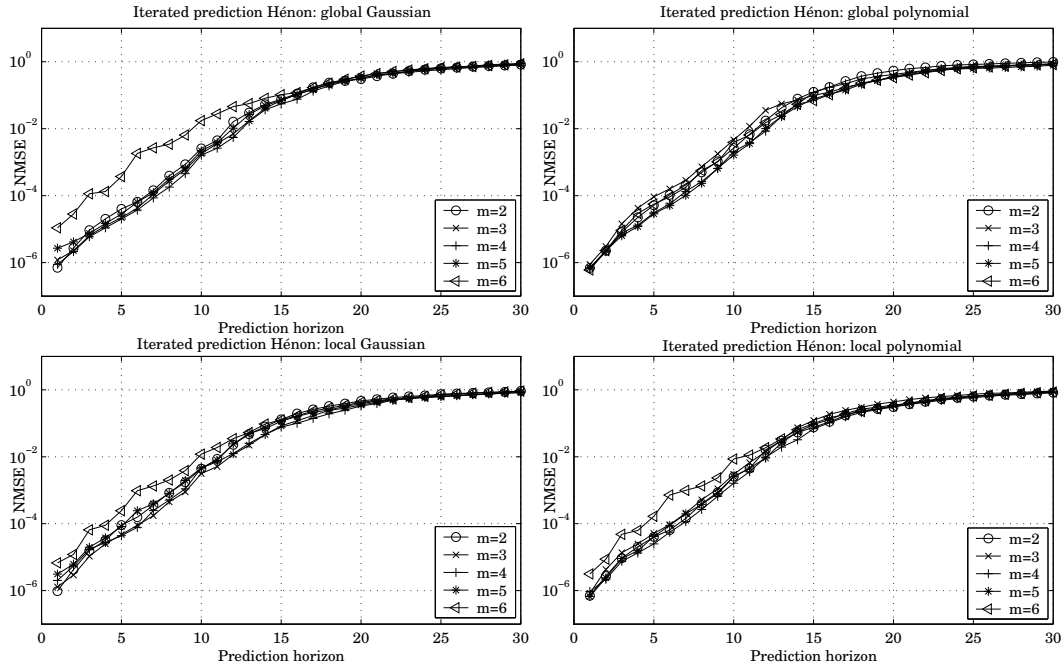


Figure 4.7: *Prediction on the Hénon time series: Iterated prediction performance of global Gaussian (top left), global polynomial (top right), local Gaussian (bottom left) and local polynomial (bottom right) SVR predictors with embedding dimensions* $m \in \{2, 3, 4, 5, 6\}$. *The iterated prediction error is averaged over 50 subsequent test sets.*

**Influence of the Embedding Dimension.** As a second result, we observe that the prediction performance gets worse with increasing embedding dimension. For global and local Gaussian predictors as well as for local polynomial ones, the worst predictor is the one having the highest embedding dimension (Table 4.2, Figure 4.9). This demonstrates the importance of the choice of the embedding dimension $m$, even in the case of noiseless data as considered here. Increasing $m$ above the minimal value necessary to unfold the attractor can make matters worse. The global polynomial predictor is less affected by the choice of $m$, and has prediction performance of the same order of magnitude for all $m$.

**Global vs. Local - Gaussian vs. Polynomial.** Although the task of learning a predictor locally is generally considered as being easier than learning a global predictor, using a local instead of a global predictor gives no improvement (in fact, it makes matters slightly worse). This may be a consequence of the relatively small training set size.

The choice of the kernel does not play an important role either. Despite of the fact that the polynomial kernel achieves slightly better single step error than the Gaussian one, the iterated prediction performance with both kernels are similar (Table 4.2). This is illustrated in Figure 4.8, where we compare the four global and local predictors using polynomial and Gaussian kernel that achieve best single step prediction performance.

**Comparison with Previously Reported Results.** In the bottom part of Table 4.2, we quote results obtained by other authors, using neural networks and RBF networks.

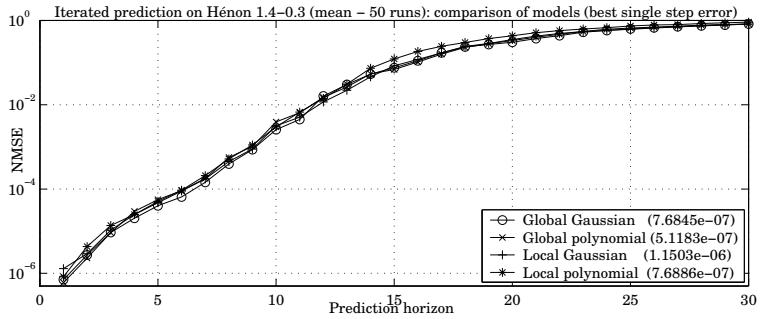


Figure 4.8: *Prediction on the Hénon time series: Comparison of iterated prediction performance of best global Gaussian, global polynomial, local Gaussian and local polynomial SVR predictors. The iterated prediction error is averaged over 50 subsequent test sets. The single step errors of the predictors are given in brackets in the legend. The predictors all have similar performance.*

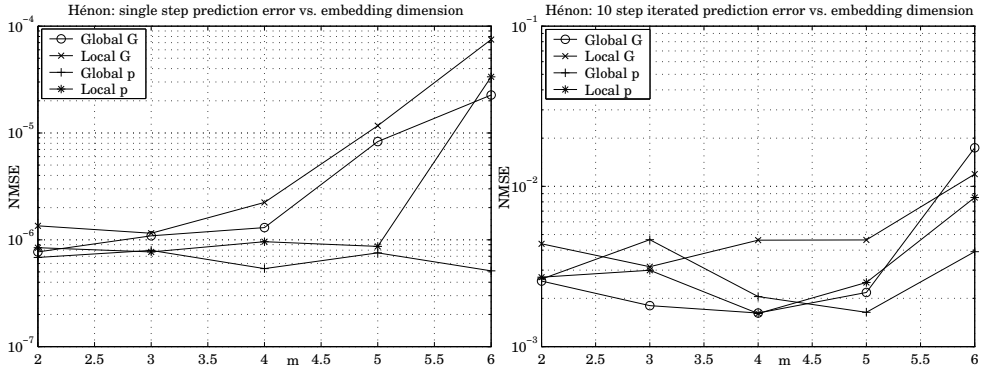


Figure 4.9: *Prediction on the Hénon time series: Influence of embedding dimension m on the prediction performance in terms of single step (left) and iterated (right) prediction error (mean on 50 subsequent test sets). The prediction performance degenerates for high embedding dimension m.*

Unfortunately, we are only aware of results in terms of single step prediction error. A comparison demonstrates that the support vector regressor approach to prediction achieves excellent performance. Even the worst SVR predictors considered have single step error of the same order of magnitude as the quoted results. This is remarkable, even more since these results are obtained using a considerably larger training set.

**The Reconstructed Attractor.** Figure 4.10 demonstrates this extraordinary performance of the SVR predictors. It shows the Hénon attractor reconstructed from 1000 samples of the true series, and compares it with the attractor reconstructed from a 1000 step iterated prediction, obtained with a global Gaussian predictor of dimension 2. We observe that the two attractors are in fact indistinguishable, indicating that the predictor was able to entirely capture the dynamics of the system. We remark that in [3], the attractor that is reconstructed from an iterated prediction, obtained with neural networks, shows considerable deformations.

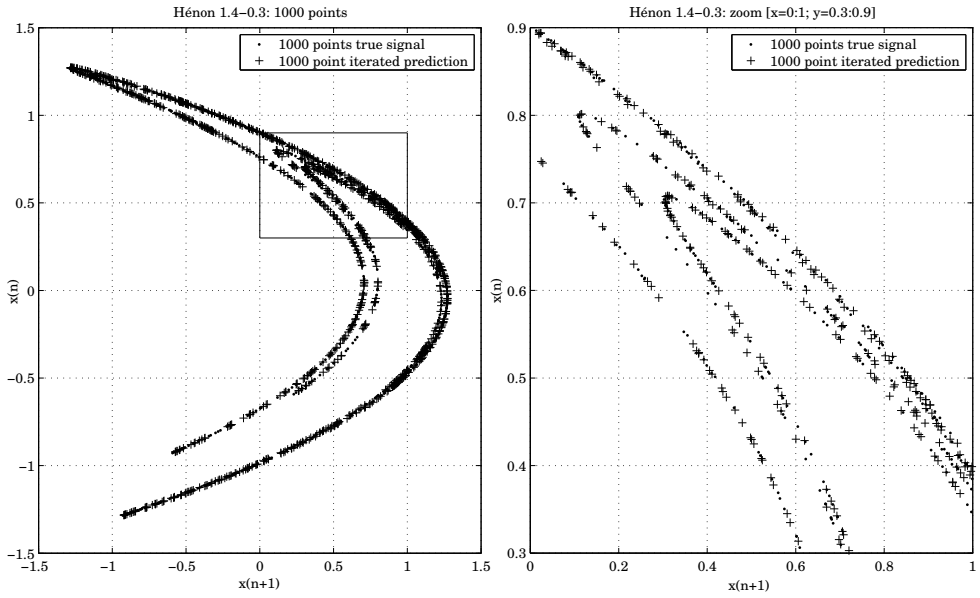


Figure 4.10: *Hénon attractor, reconstructed from the true series and from a 1000 samples iterated prediction. The two attractors are in fact indistinguishable.*

### 4.2.2 Prediction on the Mackey Glass Time Series

The $MG_{17}$ time series generated by the Mackey-Glass delay differential equation and a typical iterated prediction on it, obtained with a global Gaussian predictor, are shown in Figure 4.3 (bottom). The prediction is indistinguishable from the true series up to around prediction step 60, and keeps on staying close when iterating further. The results obtained using global and local predictors with Gaussian and polynomial kernel, averaged over 50 subsequent test sets of length 100, are given in Tables 4.3 and 4.4, together with results reported by other authors.

| Ref. | Method | $N_{\text{train}}$ | $N_{\text{run}} \cdot N_{\text{test}}$ | Single step |
|---|---|---|---|---|
| | Global G | 500 | $50 \cdot 100$ | $\mathbf{1.81 \cdot 10^{-5}}$ |
| | Local G | 500 | $50 \cdot 100$ | $1.82 \cdot 10^{-5}$ |
| | Global p | 500 | $50 \cdot 100$ | $5.52 \cdot 10^{-5}$ |
| | Local p | 500 | $50 \cdot 100$ | $1.99 \cdot 10^{-5}$ |
| [3] | Neural net | n.a. | n.a. | $1.44 \cdot 10^{-4}$ |
| [71] | Neural net | 500 | 1500 | $\mathbf{5.7 \cdot 10^{-5}}$ |

Table 4.3: *Comparison of single step prediction performance on* $MG_{17}$ *time series. All results are normalized mean squared errors. The test set and training set size are given in samples. Own results are averaged over 50 runs. "G" stands for Gaussian kernel, "p" for polynomial kernel. Best results are marked in bold.*

**Single Step Error and Iterated Prediction.** As already observed for the Hénon time series, the predictor with best single step error performance (global Gaussian) does not coincide with the predictor achieving the best iterated prediction (local Gaussian).

| Ref. | 5 step | 10 step | 20 step | 40 step | 50 step | 100 step |
|---|---|---|---|---|---|---|
| Global G | $2.63 \cdot 10^{-5}$ | $5.81 \cdot 10^{-5}$ | $4.15 \cdot 10^{-4}$ | $1.94 \cdot 10^{-3}$ | $4.64 \cdot 10^{-3}$ | $8.53 \cdot 10^{-2}$ |
| Local G | $\mathbf{1.62 \cdot 10^{-5}}$ | $3.37 \cdot 10^{-5}$ | $\mathbf{1.71 \cdot 10^{-4}}$ | $\mathbf{1.15 \cdot 10^{-3}}$ | $\mathbf{2.44 \cdot 10^{-3}}$ | $\mathbf{3.82 \cdot 10^{-2}}$ |
| Global p | $8.33 \cdot 10^{-5}$ | $2.45 \cdot 10^{-4}$ | $1.19 \cdot 10^{-3}$ | $5.17 \cdot 10^{-3}$ | $1.23 \cdot 10^{-2}$ | $1.44 \cdot 10^{-1}$ |
| Local p | $1.92 \cdot 10^{-5}$ | $\mathbf{3.22 \cdot 10^{-5}}$ | $2.78 \cdot 10^{-4}$ | $1.68 \cdot 10^{-3}$ | $3.57 \cdot 10^{-3}$ | $5.53 \cdot 10^{-2}$ |
| [3] | $\mathbf{3.24 \cdot 10^{-4}}$ | $\mathbf{3.61 \cdot 10^{-4}}$ | $\mathbf{3.61 \cdot 10^{-4}}$ | $\mathbf{1.23 \cdot 10^{-3}}$ | - | - |
| [71] | - | - | - | - | - | $\mathbf{1.8 \cdot 10^{-2}}$ |

Table 4.4: *Comparison of iterated prediction performance on* $\mathrm{MG}_{17}$ *time series. All results are normalized mean squared errors. Own results are averaged over 50 runs. "G" stands for Gaussian kernel, "p" for polynomial kernel. Best results are marked in bold.*

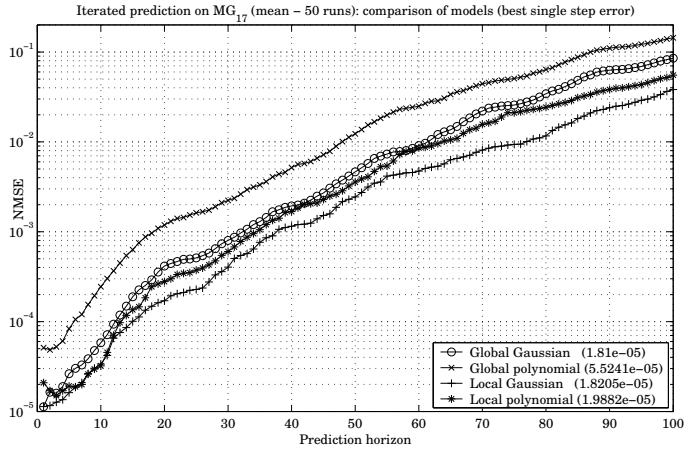


Figure 4.11: *Prediction on the Mackey-Glass time series: Comparison of SVR predictor performance (mean on 50 runs). The single step prediction errors are given in brackets in the legend. The local Gaussian predictor outperforms all other predictors.*

For the $MG_{17}$ series, however, the difference in single step error for these two predictors is very small. The SVR predictor with worst single step error (global polynomial) has again worst iterated prediction performance.

**Global vs. Local - Gaussian vs. Polynomial.** We observe further that the choice of kernel plays an important role for predicting the Mackey-Glass time series. For both global and local predictors, the Gaussian kernel outperforms the polynomial kernel.

What is more, the local predictors achieve considerably better results than the global ones, despite of the small training set. These facts are illustrated in Figure 4.11, comparing the single step and iterated prediction performance of all four SVR predictors considered.

**Comparison with Previously Reported Results.** The comparison with results obtained by other authors, using neural networks, demonstrates the good performance of the SVR predictors on this time series. The prediction error for single step prediction and iterated prediction up to iteration 40 is lower than any result we are aware of, and the 100 step iterated prediction is comparable with the one obtained in [71]. Notice that

we use the same training set size as in [71], and that the training and test set size is not given in [3].

### 4.2.3 Prediction on the Lorenz Time Series

We consider the series given by the x-coordinate of the Lorenz attractor, obtained by numerical integration with integration step $h = 0.01$ and $h = 0.05$. Typical predictions on both series are shown in Figure 4.12. We observe that the iterated predictions lose track of the true trajectories after approximately 40 prediction steps, both at a point where the series reaches a value of $x \approx 0$. This region of the state space is critical, since here the evolution of the states is very fast.

In Tables 4.5 and 4.6, we compare the performance of local and global predictors with Gaussian and polynomial kernel, averaged over 50 subsequent test sets.

| Ref. | Method | Int. Step | $N_{\text{train}}$ | $N_{\text{run}} \cdot N_{\text{test}}$ | Single step |
|---|---|---|---|---|---|
| | Global G | h=0.01 | 1000 | $50 \cdot 100$ | $1.13 \cdot 10^{-4}$ |
| | Local G | | | $50 \cdot 100$ | $2.86 \cdot 10^{-4}$ |
| | Global p | | | $50 \cdot 100$ | $\mathbf{7.27 \cdot 10^{-5}}$ |
| | Local p | | | $50 \cdot 100$ | $1.39 \cdot 10^{-4}$ |
| | Global G | h=0.05 | 1000 | $50 \cdot 100$ | $\mathbf{3.12 \cdot 10^{-6}}$ |
| | Local G | | | $50 \cdot 100$ | $3.30 \cdot 10^{-6}$ |
| | Global p | | | $50 \cdot 100$ | $2.63 \cdot 10^{-5}$ |
| | Local p | | | $50 \cdot 100$ | $3.81 \cdot 10^{-6}$ |
| [40] | Local average | n.a. | 3000 | - | $6.76 \cdot 10^{-4}$ |
| [71] | Neural net | n.a. | 4000 | 1500 | $\mathbf{3.4 \cdot 10^{-5}}$ |

Table 4.5: *Comparison of single step prediction performance on Lorenz time series. All results are normalized mean squared errors. The test set and training set size are given in samples. Own results are averaged over 50 runs. "G" stands for Gaussian kernel, "p" for polynomial kernel. Best results are marked in bold.*

| Ref. | Int. Step | 5 step | 10 step | 20 step | 40 step | 50 step |
|---|---|---|---|---|---|---|
| Global G | h=0.01 | $4.37 \cdot 10^{-4}$ | $2.05 \cdot 10^{-3}$ | $1.04 \cdot 10^{-2}$ | $2.35 \cdot 10^{-1}$ | 1.01 |
| Local G | | $\mathbf{1.66 \cdot 10^{-4}}$ | $\mathbf{9.29 \cdot 10^{-4}}$ | $\mathbf{5.88 \cdot 10^{-3}}$ | $8.31 \cdot 10^{-2}$ | $\mathbf{1.20 \cdot 10^{-1}}$ |
| Global p | | $3.17 \cdot 10^{-4}$ | $1.94 \cdot 10^{-3}$ | $1.38 \cdot 10^{-2}$ | $2.80 \cdot 10^{-1}$ | $7.11 \cdot 10^{-1}$ |
| Local p | | $1.95 \cdot 10^{-4}$ | $1.26 \cdot 10^{-3}$ | $7.57 \cdot 10^{-2}$ | $\mathbf{4.53 \cdot 10^{-2}}$ | $3.89 \cdot 10^{-1}$ |
| Global G | h=0.05 | $1.77 \cdot 10^{-4}$ | $3.67 \cdot 10^{-4}$ | $8.07 \cdot 10^{-4}$ | $5.64 \cdot 10^{-2}$ | $9.21 \cdot 10^{-2}$ |
| Local G | | $\mathbf{5.07 \cdot 10^{-5}}$ | $\mathbf{1.60 \cdot 10^{-4}}$ | $\mathbf{5.04 \cdot 10^{-4}}$ | $\mathbf{4.03 \cdot 10^{-2}}$ | $8.02 \cdot 10^{-2}$ |
| Global p | | $2.02 \cdot 10^{-3}$ | $1.40 \cdot 10^{-2}$ | $2.16 \cdot 10^{-1}$ | $6.46 \cdot 10^{-1}$ | $NaN$ |
| Local p | | $6.85 \cdot 10^{-5}$ | $2.22 \cdot 10^{-4}$ | $8.58 \cdot 10^{-4}$ | $5.90 \cdot 10^{-2}$ | $\mathbf{7.68 \cdot 10^{-2}}$ |
| [40] | n.a. | $\mathbf{8.41 \cdot 10^{-4}}$ | $\mathbf{1.09 \cdot 10^{-3}}$ | $\mathbf{3.25 \cdot 10^{-3}}$ | $\mathbf{3.03 \cdot 10^{-3}}$ | - |

Table 4.6: *Comparison of iterated prediction performance on Lorenz time series. All results are normalized mean squared errors. Own results are averaged over 50 runs. "G" stands for Gaussian kernel, "p" for polynomial kernel. Best results are marked in bold.*

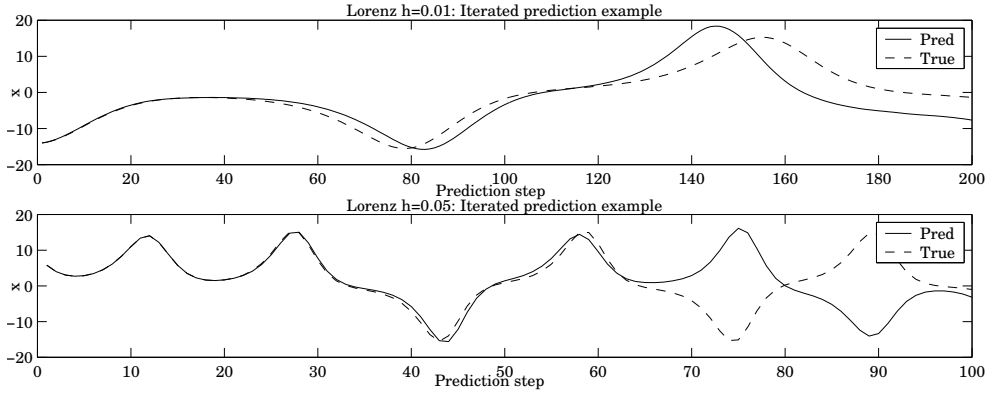


Figure 4.12: *Typical predictions on the Lorenz time series for integration time step* $h = 0.01$ *(top) and* $h = 0.05$ *(bottom).*

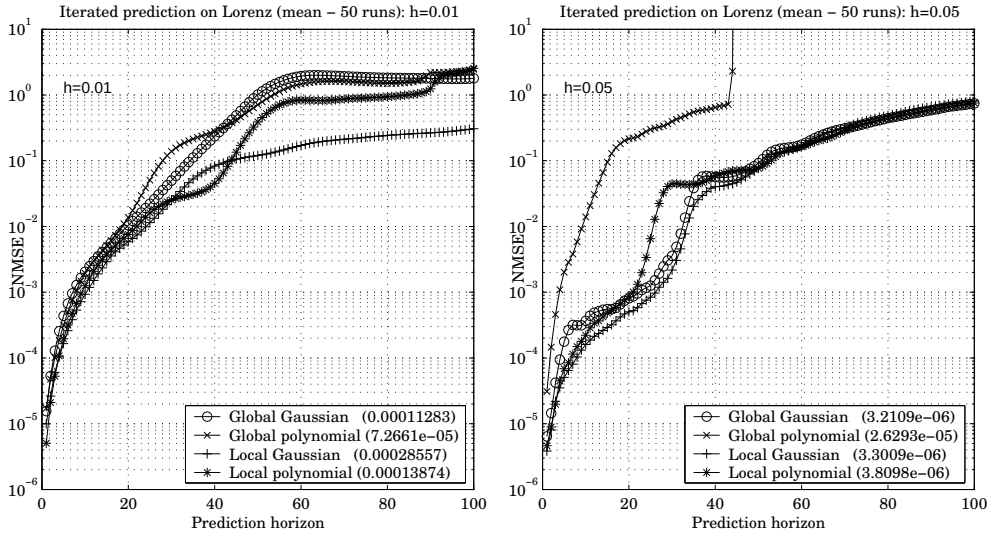


Figure 4.13: *Prediction on the Lorenz time series: Comparison of performance of all SVR predictors (mean on 50 subsequent test sets).*

**Influence of the Integration Step.** The first thing we note is that the prediction performance depends crucially on the integration step $h$. Unfortunately, in many of the literature considering prediction of observations of the Lorenz attractor, the integration step is not stated, making comparisons difficult. Nevertheless, we can observe qualitative similarities in the prediction performance on both series, and that the prediction performance is comparable to that reported by other authors, employing local averaging techniques and neural networks.

**Single Step Error and Iterated Prediction.** As already observed for the Hénon and the $MG_{17}$ time series, the single step prediction error measure is not sufficient to describe the performance of the predictors. The SVR predictor with best single step error does not have best iterated prediction performance. For $h = 0.01$, it is in fact the predictor with worst single step error that has best iterated prediction performance, whereas for

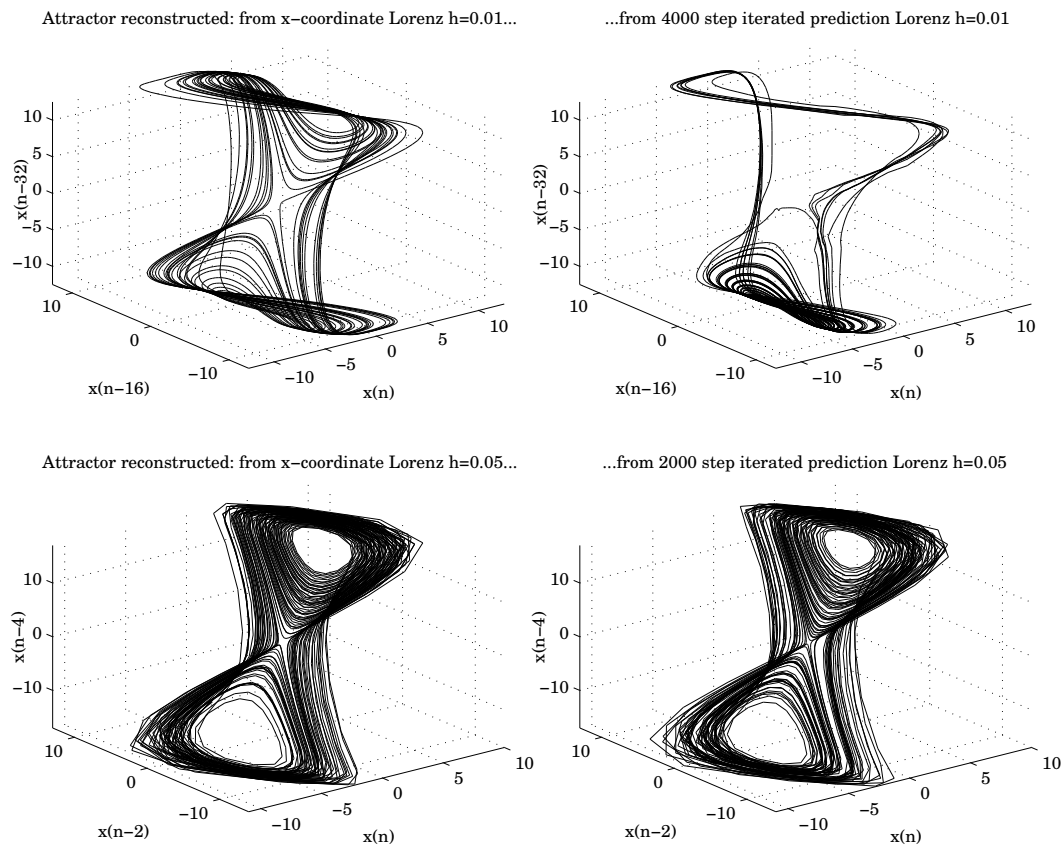


Figure 4.14: *Attractors reconstructed from the real Lorenz time series (left column), and from iterated predictions (right column) for integration time step* $h = 0.01$ *(top) and* $h = 0.05$ *(bottom). For* $h = 0.01$*, the prediction does not explore the whole state space, whereas for* $h = 0.05$*, the attractors reconstructed from the prediction and from the real series are similar.*

$h = 0.05$ worse single step error coincides with worse iterated prediction.

**Global vs. Local - Gaussian vs. Polynomial.** We note further that for both series, the global predictors perform worse than the local ones, and predictors employing a Gaussian kernel are better than those using a polynomial kernel. This can as well be seen from Figure 4.13, where we compare the prediction performance of all four SVR predictors. The local Gaussian predictor outperforms all others for both series considered.

What is more, we are even unable to obtain a stable global predictor when using a polynomial kernel for $h = 0.05$. The prediction and the prediction error grow unboundedly after 44 prediction steps. Note, however, that the local model with polynomial kernel is stable (Figure 4.13, right).

**The Reconstructed Attractors.** In Figure 4.14, the attractors in embedding space, reconstructed from the x-coordinate of the Lorenz system, are shown. On the left, the attractor is reconstructed from the true signal, whereas on the right, an iterated prediction of length 4000 ($h = 0.01$) and 2000 ($h = 0.05$) samples, obtained with a local Gaussian

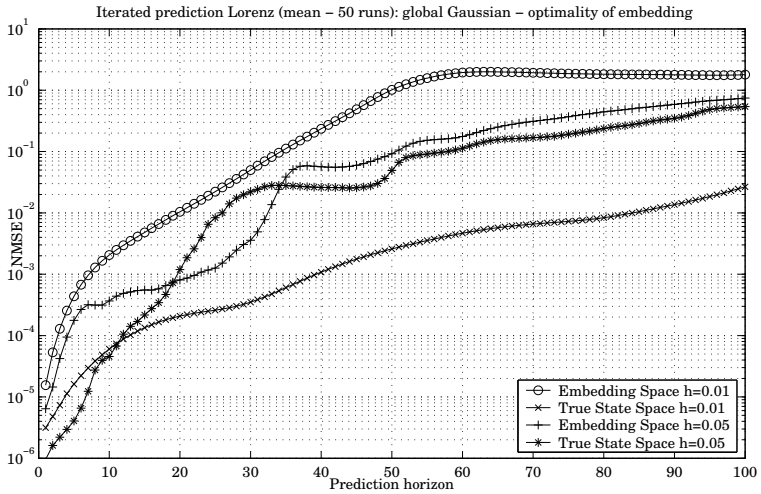


Figure 4.15: *Prediction on the Lorenz time series: Comparison of prediction performance of global Gaussian SVR predictors working in embedding space and in true state space. For integration time step $h = 0.01$, the difference in prediction performance between prediction in true state space and in embedding space is very large (up to more than two orders of magnitude), whereas it is less pronounced for $h = 0.05$.*

model, is used. We observe that for $h = 0.01$, the prediction at first starts exploring the phase space, but gets then stuck in a quasi-periodic motion in the lower part of the attractor. The predictor is therefore not able to correctly approximate the dynamics of the system. In contrast, for $h = 0.05$, the iterated prediction continues exploring the whole state space, and its reconstructed attractor is similar to that reconstructed from the true signal. This indicates that in this case, the dynamics are well approximated by the predictor.

**Optimal Choice of the Embedding Parameters.** The fact that the support vector predictors are unable to properly learn the dynamics of the system for $h = 0.01$ is at least partly due to a non-optimal embedding. This can be seen from Figure 4.15, where we compare the prediction performance of the global Gaussian predictor working in embedding space (as considered above) with the prediction performance of a global Gaussian predictor working in the true state space of the Lorenz attractor[10]. We observe that for $h = 0.01$, the support vector predictors working in the true state space perform considerably better than the predictor trained in the embedding space. For $h = 0.05$, the difference in performance is less pronounced.

Notice as well the difference in prediction performance for the predictions obtained in true state space for the two integration time steps considered, demonstrating again the influence of the integration time step.

[10]This prediction is obtained by training one SVR predictor per coordinate.

### 4.2.4 Prediction on Santa Fe Data Set A

The experimental Santa Fe data set A is a famous benchmark set for chaotic time series prediction. Many authors have considered this time series in their work on time series prediction, which together with the fact that the training set and test set are prescribed allows for thorough comparison.

Our results are summarised in Tables 4.7 and 4.8, together with results reported by other authors. The predictions we obtain on the five test sets are given in Figure 4.16.

| Ref. | Test set: 1001-1100 | 2181-2280 | 3871-3970 | 4001-4100 | 5181-5280 |
|---|---|---|---|---|---|
| Global G | $\mathbf{9.20 \cdot 10^{-3}}$ | $\mathbf{1.46 \cdot 10^{-2}}$ | $\mathbf{1.66 \cdot 10^{-2}}$ | $2.03 \cdot 10^{-4}$ | $1.32 \cdot 10^{-2}$ |
| Local G | $1.57 \cdot 10^{-2}$ | $1.85 \cdot 10^{-2}$ | $1.77 \cdot 10^{-2}$ | $2.48 \cdot 10^{-4}$ | $1.98 \cdot 10^{-2}$ |
| Global p | $1.37 \cdot 10^{-2}$ | $2.29 \cdot 10^{-2}$ | $9.49 \cdot 10^{-2}$ | $\mathbf{1.93 \cdot 10^{-4}}$ | $1.38 \cdot 10^{-2}$ |
| Local p | $1.02 \cdot 10^{-2}$ | $2.17 \cdot 10^{-2}$ | $6.49 \cdot 10^{-2}$ | $2.25 \cdot 10^{-4}$ | $\mathbf{1.31 \cdot 10^{-2}}$ |
| [47] | $6.3 \cdot 10^{-2}$ | - | - | - | - |
| [71] | $2.76 \cdot 10^{-3}$ | - | - | - | - |
| [72] | $\mathbf{2.3 \cdot 10^{-2}}$ | - | - | - | - |

Table 4.7: *Comparison of single step prediction performance on Santa Fe data set A. All results are normalized mean squared errors. "G" stands for Gaussian kernel, "p" for polynomial kernel. Best results are marked in bold.*

| Ref. | Test set: 1001-1100 | 2181-2280 | 3871-3970 | 4001-4100 | 5181-5280 |
|---|---|---|---|---|---|
| Global G | $\mathbf{4.46 \cdot 10^{-2}}$ | $3.93 \cdot 10^{-1}$ | $4.31 \cdot 10^{-1}$ | $8.89 \cdot 10^{-4}$ | $\mathbf{4.81 \cdot 10^{-2}}$ |
| Global p | $5.74 \cdot 10^{-1}$ | $NaN$ | $NaN$ | $\mathbf{1.81 \cdot 10^{-4}}$ | $NaN$ |
| Local G | $2.82 \cdot 10^{-1}$ | $\mathbf{5.16 \cdot 10^{-2}}$ | $\mathbf{3.06 \cdot 10^{-1}}$ | $1.11 \cdot 10^{-2}$ | $8.33 \cdot 10^{-2}$ |
| Local p | $5.31 \cdot 10^{-1}$ | $1.88 \cdot 10^{-1}$ | $3.73 \cdot 10^{-1}$ | $1.61 \cdot 10^{-2}$ | $8.27 \cdot 10^{-1}$ |
| [47] | 1.03 | - | - | - | - |
| [17] | $2.6 \cdot 10^{-2}$ | - | - | - | - |
| [14] | $2.47 \cdot 10^{-1}$ | - | - | - | - |
| [4] | $2.8 \cdot 10^{-2}$ | $5.1 \cdot 10^{-2}$ | $2.55 \cdot 10^{-1}$ | $3.9 \cdot 10^{-2}$ | - |
| [4]-CV | $2.9 \cdot 10^{-2}$ | $\mathbf{2.8 \cdot 10^{-2}}$ | $\mathbf{3.0 \cdot 10^{-3}}$ | $3.0 \cdot 10^{-2}$ | - |
| [71] | $\mathbf{1.94 \cdot 10^{-2}}$ | - | - | - | - |
| [57] | $6.6 \cdot 10^{-2}$ | $6.1 \cdot 10^{-2}$ | $8.6 \cdot 10^{-2}$ | $4.79 \cdot 10^{-1}$ | $\mathbf{3.8 \cdot 10^{-2}}$ |
| [50] | $7.7 \cdot 10^{-2}$ | $1.74 \cdot 10^{-1}$ | $1.83 \cdot 10^{-1}$ | $\mathbf{6.0 \cdot 10^{-3}}$ | $1.11 \cdot 10^{-1}$ |
| [72] | $2.73 \cdot 10^{-2}$ | $6.5 \cdot 10^{-2}$ | $4.87 \cdot 10^{-1}$ | $2.3 \cdot 10^{-2}$ | $1.6 \cdot 10^{-1}$ |

Table 4.8: *Comparison of 100 step iterated prediction performance Santa Fe data set A. All results are normalized mean squared errors. "G" stands for Gaussian kernel, "p" for polynomial kernel. Best results are marked in bold.*

**Global vs. Local - Gaussian vs. Polynomial.** We observe that none of the predictors wins on all five test sets, in terms of neither single step prediction error nor iterative prediction error. This is consistent with results reported by other authors, where no predictor performs particularly well on all five test sets simultaneously. For the predictors we consider, this is especially obvious for the global polynomial one, which has worst iterated

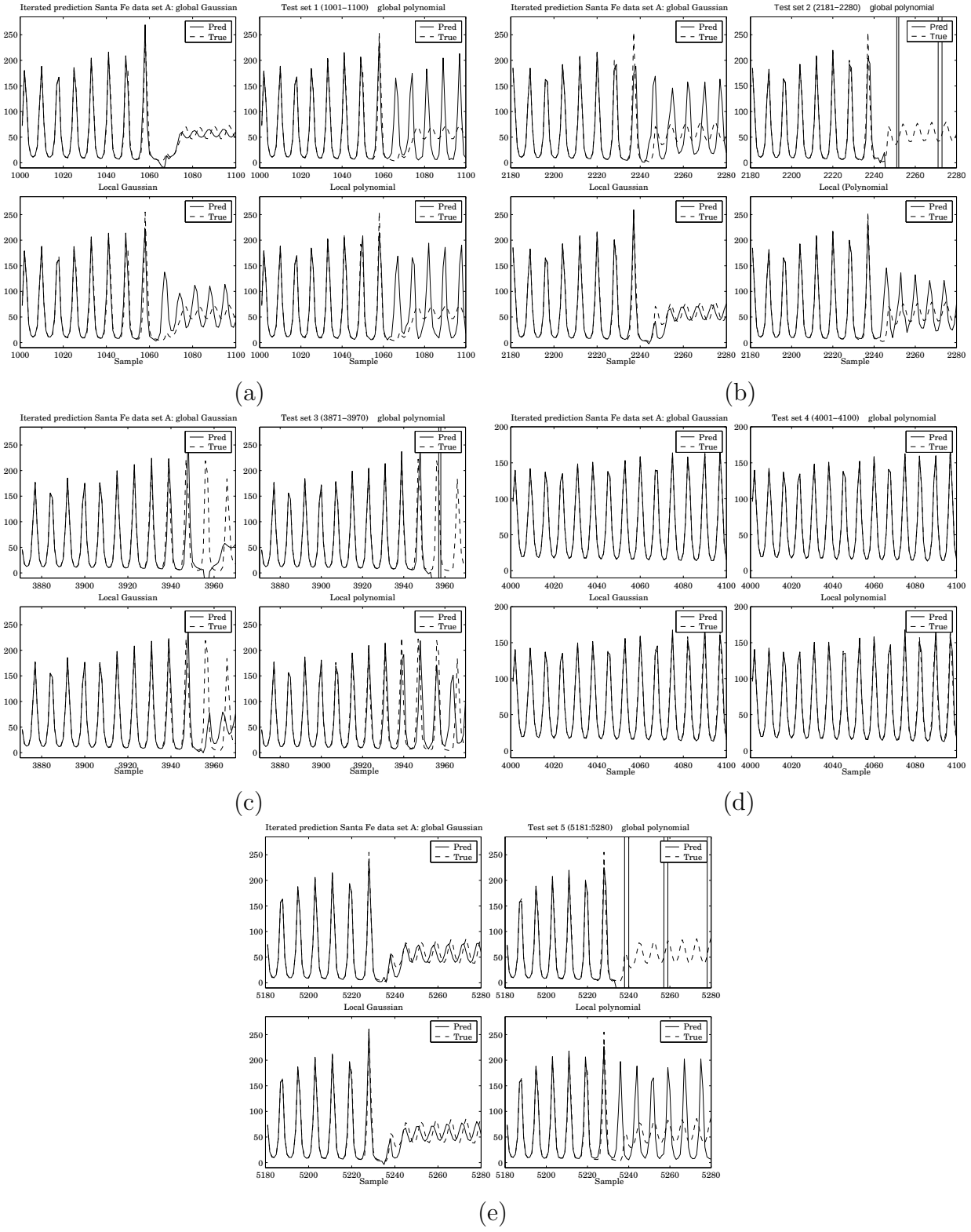


Figure 4.16: *Prediction on Santa Fe data set A: test set 1 (a), test set 2 (b), test set 3 (c), test set 4 (d), test set 5 (e). In the subfigures, the predictions on the particular test set obtained with global Gaussian (top left), global polynomial (top right), local Gaussian (bottom left) and local polynomial (bottom right) SVR predictors are shown.*

| Ref. | Method |
|---|---|
| [47] | Linear dynamical model in kernel space |
| [17] | Kernel recursive least squares algorithm |
| [14] | Vector quantization |
| [4] | Local regression approach |
| [4]-CV | Local regression approach with cross validation |
| [71] | Neural network |
| [57] | Neural network |
| [50] | Local linear model |
| [72] | Neural network |

Table 4.9: *Prediction methods employed in the quoted literature.*

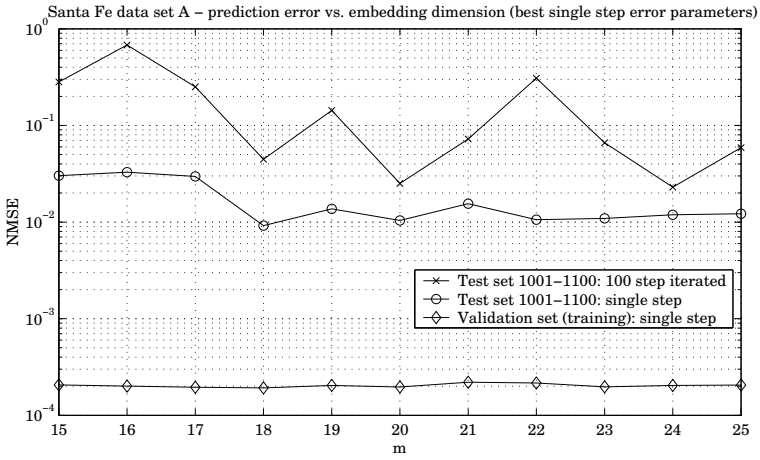


Figure 4.17: *Santa Fe data set A: Prediction performance of global Gaussian SVR predictors on validation set (samples* $901 - 1000$*) and test set 1 (samples* $1001 - 1100$*), for different embedding dimensions* $m \in \{15, \cdots, 25\}$*. The predictor in embedding space with dimension* $m = 18$ *has lowest single step prediction error on the validation set and on the test set. The predictor with lowest iterated prediction error on the test set works in embedding space of dimension* $m = 24$*.*

prediction performance for all test sets except test set 4, for which it is not only the best among all predictors we consider, but as well outperforms the best predictors reported in literature we are aware of. For three of the test sets the global polynomial predictor does, however, not even produce a stable prediction.

Nevertheless, we note that all in all predictors with Gaussian kernels, and especially the global Gaussian one, produce better predictions than the predictors employing a polynomial kernel for this data set. The global Gaussian predictor produces the best or at least close to best prediction on all five test sets in terms of single step error, and it has the best or close to best 100 step iterated prediction performance on all test sets except test set 3.

Santa Fe data set A is considered as difficult to predict, since prediction involves forecasting the occurrence of a collapse in three of the five test sets from only one example of such a collapse in the training set. We remark that the global as well as the local

Gaussian SVR predictors are able to forecast these occurrences in all but one cases, whereas the predictors using polynomial kernels completely fail to forecast the collapses (Figure 4.16).

**Comparison with Previously Reported Results.** The global Gaussian predictor has lowest single step prediction error on the validation set (samples $901 - 1000$). Therefore, it is selected for comparison with other results reported in literature. Notice that for some of these results, the series has been pre-processed. For instance, in [50] the series is denoised with some FFT technique and upsampled. Kohlmorgen claims in [31] that this renders the prediction task easier. We obtain our predictions directly on the series as it is, without any extra technique. The prediction methods employed in the quoted literature are summarised in Table 4.9.

The global Gaussian predictor achieves performance one order of magnitude better than the best result we are aware of on test set 4. It is, however, 2 orders of magnitude worse than the best result reported on test set 3. For all other test sets, it has performance comparable to the state of the art.

**Influence of the Embedding Dimension.** In Figure 4.17, we investigate the influence of the embedding dimension on the prediction performance on test set 1 for global Gaussian predictors. Clearly, the predictor of dimension 18 has lowest single step error on both the validation set and the training set. The best 100 step iterated prediction is, however, obtained with a predictor of dimension 24. This suggests that our results might be improvable by choosing a different embedding dimension.

### 4.2.5 Discussion

We consider predicting three synthetic and one experimental time series exhibiting chaotic behaviour, using global and local support vector regressors of type (3.11-3.13) and (3.14-3.15), respectively, with Gaussian and polynomial kernels, as predictors. We observe that none of the predictors is superior to the others for all data sets and performance measures considered. The best predictors we find for each data set, however, achieve excellent performance, especially for the Hénon series, the $MG_{17}$ series, and Santa Fe Data Set A.

**Global vs. Local Model.** A local predictor for a chaotic time series is in general less complex and therefore easier to learn than a global one, since the prediction function does not have to approximate the dynamics of the system for the entire state space. Therefore, local predictors are generally believed to achieve better prediction performance [32]. Nevertheless, the best predictors for two of the time series considered (Hénon and Laser) are global. These predictors both employ Gaussian kernels, which in themselves possess already good localisation properties. A second reason for this unexpected result may be the fact that we consider small training sets, so that the state space and the attractor are only loosely sampled. In this case, a local predictor has to produce an approximation of the evolution of a state from a small set of neighbouring states that may be considerably far away in state space, and therefore may have little to say about the evolution of the state under consideration. In contrast, if the training set grows large enough, and the

state space is sampled tightly, the neighbours of a state contribute enough information about the local dynamics of the state to allow for a good local approximation. What is more, when considering local models, an additional parameter - the number of nearest neighbours - has to be determined. For that, no theoretical rule exists, and the empirically selected number of nearest neighbours may be suboptimal.

Independently of considerations about prediction performance, the local method has computational advantages for large training set sizes. The global predictor has to be found on the entire training set, whereas the local predictor is trained on only a small number of samples. Although a new local predictor has to be found for each single prediction iteration, the local prediction may therefore be obtained at less computational cost than a prediction with a global model.

**Gaussian vs. Polynomial Kernel.** In all cases considered, polynomial kernels are outperformed by Gaussian kernel in terms of iterated prediction performance. What is more, some of the predictors employing polynomial kernels become unstable, producing an iterated prediction that grows unboundedly (Lorenz time series and data set A). This may be due to the fact that Gaussian kernels possess good localisation properties, such that states far away from the state we are currently predicting from contribute nearly nothing to the prediction if the kernel width is chosen properly. The prediction cannot grow unboundedly for a Gaussian kernel, since the value of the kernel function decreases fast for two embedding vectors that are far from the support vectors[11].

Polynomial kernels do not posses this property and can therefore produce predictions that grow unboundedly. Nevertheless, in some cases the best single step prediction error is achieved with a polynomial kernel.

**Prediction Performance Measures.** We observe that the single step prediction error cannot completely describe the quality of a predictor. Specifically, in many cases it does not allow for statements about the iterated prediction performance (Hénon, $MG_{17}$, and Lorenz time series). This result may seem surprising at the first moment, since a predictor with lower single step error introduces less errors per prediction step. The single step error can, however, not measure how these errors propagate when the prediction is iterated. A model may produce better single step predictions and may at the same time be more sensitive to the error that is introduced at each prediction step. Then, the initially higher accuracy of the prediction, as compared with the accuracy of a predictor with larger single step error but less sensitivity, may be lost after a few iteration steps.

**Simulation of Chaotic Systems.** When numerical simulations of chaotic systems are required for obtaining a time series, interpretations of the results have to be taken with care. This has already been discussed in Subsection 4.1.1, and has been confirmed in our results on the Lorenz time series. The prediction performance on the two series obtained from the same system of equations with a different integration time step is far from being equivalent.

[11] If the kernel width is chosen too narrow, it can therefore happen that the predictor output converges to zero.

**Optimal Choice of the Embedding Parameters.** The choice of an appropriate embedding dimension is especially important in the presence of noise, since choosing an embedding dimension that is larger than needed for unfolding the orbits unnecessarily amplifies the noise. It is, however, observed that the embedding dimension influences the prediction performance as well in the noiseless case (Hénon time series). What is more, in one of the experiments the embedding was not chosen correctly, which significantly degenerated the prediction performance potentially achievable with the function estimation technique used (Lorenz time series with $h = 0.01$).

**Optimal Choice of the Hyperparameters.** Due to the determination of the SVM hyperparameters through an empirical validation technique, some tendencies in the results obtained may be hidden. Although a considerable effort has been undertaken to determine good hyperparameters, it may not have been possible to find the optimal parameter setting in all cases.

# Summary and Outlook

In this work, we considered SVMs as a nonparametric supervised learning method for function estimation. SVMs are based on results from statistical learning theory, optimisation theory and functional analysis. They choose their modelling function from a rich function base determined by a kernel and implement an induction principle that allows them to control the capacity of these modelling function. They can therefore avoid overfitting and address the curse of dimensionality. Moreover, the solution to the training algorithm is unique and can be found efficiently as the solution of a quadratic programme.

The contribution of this work can be found in two aspects. First, in Chapter 2, we gave an introduction to the theory of SVMs that is accessible for the reader that is not familiar with this subject, and provided an outlook on more advanced topics. Introductory literature bringing together all necessary theory for the development of SVMs is hard to find, since this approach is still subject to active research. We considered in detail the derivation of standard SVMs for regression estimation and established their connection to other function estimation methods, and reviewed the important question of hyperparameter selection. The chapter covers all theory necessary for understanding and applying SVMs.

Second, we gave an exhaustive comparison of results for the specific task of chaotic time series prediction. For that, we briefly introduced the topic of chaos in dynamical systems and discussed how SVMs can be used to approximate a prediction function. We have brought together results reported in literature and demonstrated the excellent performance of SVMs for this specific task.

SVMs report excellent performance for a wide range of applications. In contrast to neural networks, where an appropriate network architecture has to be elaborated by the user for each specific application, and where the training procedure may be time consuming and affected by local minima, the SVM algorithm is fast, chooses the appropriate architecture inline according to results from statistical learning theory, and leaves only very few so-called hyperparameters, namely the trade-off parameter $C$, the width of the $\epsilon$-insensitive zone, and the kernel parameters, to be tuned by the user. SVMs could therefore be thought of as an excellent tool as well for non-expert users. The choice of the hyperparameters is, however, not trivial, and restricts some of the flexibility and practical advantages inherent in this approach. Further work will be necessary to develop efficient principles for hyperparameter selection and to make the computationally costly and suboptimal empirical selection techniques obsolete. Still, SVMs stay an extremely powerful and advantageous learning machine with excellent empirical performance.

# Appendix A

# Abbreviations

| **Abbreviation** | |
|---|---|
| dde | delay differential equation |
| ERM | empirical risk minimisation |
| FNN | false nearest neighbors |
| GACV | generalised approximate cross validation |
| HO testing | hold out testing |
| kCV | k-fold cross validation |
| KKT complementary conditions | Karush Kuhn Tucker complementary conditions |
| LOO testing | leave one out testing |
| MAP | maximum a posterior |
| $MG_{17}$ | Mackey-Glass time series with delay $T = 17$ |
| NN | neural network, nearest neighbours |
| RBF | radial basis function |
| RKHS | reproducing kernel Hilbert space |
| SRM | structural risk minimisation |
| SVC | support vector machine for classification |
| SVM | support vector machine |
| SVR | support vector machine for regression estimation |
| VC dimension | Vapnik-Chervonenkis dimension |

# Bibliography


[1] H.D.I. Abarbanel, *Analysis of observed chaotic data*, first ed., Springer, New York, 1996.

[2] P.J. Angeline, *Evolving predictors for chaotic time series*, Proceedings of SPIE: Application and Science of Computational Intelligence (S. Rogers, D. Fogel, J. Bezdek, and B. Bosacchi, eds.), vol. 3390, 1998, pp. 170–80.

[3] A. Aussem, *Dynamical recurrent neural networks towards prediction and modeling of dynamical systems*, Neurocomputing **28** (1999), no. 3, 207–232.

[4] G. Bontempi and M. Birattari, *A multi-step-ahead prediction method based on local dynamic properties*, ESANN 2000 Proceedings - European Symposium on Artificial Neural Networks, 2000, pp. 311–316.

[5] C. J. Burges, *A tutorial on support vector machines for pattern recognition*, 1998.

[6] L. Cao, *Support vector machines experts for time series forecasting*, Neurocomputing **51** (2003), 321–339.

[7] E. Castillo, J. M. Gutierrez, A. Cobo, and C. Castillo, *A minimax method for learning functional networks*, Neural Process. Lett. **11** (2000), no. 1, 39–49.

[8] G. Cauwenberghs and T. Poggio, *Incremental and decremental support vector machine learning*, Advances in Neural Information Processing Systems (NIPS 2000) **13** (2001), 409–415.

[9] A. Chalimourda, B. Schölkopf, and A. Smola, *Experimentally optimal $\nu$ in support vector regression for different noise models and parameter settings*, Neural Networks **17 (1)** (2004), 127–141.

[10] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, *Choosing multiple parameters for support vector machines*, Machine Learning **46 (1-3)** (2002), 131–159.

[11] V. Cherkassky and Y. Ma, *Practical selection of svm parameters and noise estimation for svm regression*, Neural Networks **17 (1)** (2004), 113–126.

[12] C. Cortes and V. Vapnik, *Support vector networks*, Machine Learning **20** (1995), 1–25.

[13] N. Cristianini and J. Shawe-Taylor, *An introduction to support vector machines and other kernel-based learning methods*, Cambridge University Press, 2000.

[14] G. Dangelmayr, S. Gadaleta, D. Hundley, and M. Kirby, *Time series prediction by estimating markov probabilities through topology preserving maps*, Proc. SPIE - Applications and Science of Neural Networks, Fuzzy Systems, and Evolutionary Computation II (B. Bosacchi, D. B. Fogel, and J.C. Bezdek, eds.), vol. 3812, 1999, pp. 86–93.

[15] K. Duan, S. Keerthi, and A. Poo, *Evaluation of simple performance measures for tuning svm hyperparameters*, Neurocomputing **51** (2003), 41–59.

[16] S.V. Dudul, *Prediction of a lorenz chaotic attractor using two-layer perceptron neural network*, Applied Soft Computing (2004), To appear.

[17] Y. Engel, S. Mannor, and R. Meir, *The kernel recursive least-squares algorithm*, IEEE Transaction on Signal Processing **52** (2004), no. 8, 2275–2285.

[18] J.D. Farmer and J.J. Sidorowich, *Predicting chaotic time series*, Physical Review Letters **59** (1987), no. 8, 845–848.

[19] R. Fernandez, *Predicting time series with a local support vector regression machine*, 1999.

[20] R. Fletcher, *Practical methods of optimization*, second ed., John Wiley & Sons Ltd., Chichester, 1987.

[21] F. A. Gers, D. Eck, and J. Schmidhuber, *Applying LSTM to time series predictable through time-window approaches*, Lecture Notes in Computer Science **2130** (2001), 669+.

[22] P. Grassberger and I. Procaccia, *Measuring the strangeness of strange attractors*, Physica D **9** (1983), 189–208.

[23] S. Gunn, *Support vector machines for classification and regression*, Tech. report, Department of Electronics and Computer Science, University of Southampton, 1998.

[24] M. Han, J. Xi, and F. Yin, *Prediction of chaotic time series based on the recurrent predictor neural network*, IEEE Transactions on Signal Processing **52** (2004), no. 12, 3409–3416.

[25] U. Hübner, C.O. Weiss, N. B. Abraham, and D. Tang, *Lorenz-like chaos in $nh_3$-fir lasers (data set a)*, Time Series Prediction: Forecasting the Future and Understanding the Past (A.S. Weigend and N.A. Gershenfeld, eds.), Addison-Wesley, 1994, pp. 73–104.

[26] R. Hegger, H. Kantz, and T. Schreiber, *Practical implementation of nonlinear time series methods: The tisean package*, Chaos **9** (1999), no. 2, 413–435.

[27] M. Henon, *A two-dimensional mapping with a strange attractor*, Comm.Math.Phys. **50** (1976), no. 1, 69–77.

[28] P. Huber, *Robust estimation of location parameter*, Annals of Mathematical Statistics **35** (1964), 73–101.

[29] T. Joachims, *Making large-scale svm learning practical*, Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.) (1999), 169–184.

[30] ———, *The maximum-margin approach to learning text classifiers: method, theory and algorithms*, Ph.D. thesis, University of Dortmund, 2001.

[31] J. Kohlmorgen and K.-R. Müller, *Data set a is a pattern matching problem*, Neural Process. Lett. **7** (1998), no. 1, 43–47.

[32] D. Kugiumtzis, B. Lillekjendlie, and N. Christophersen, *Chaotic time series: Part i. estimation of invariant properties in state space*, Modeling, Identification and Control **15** (1994), no. 4, 205–224.

[33] J. Kwok and I. Tsang, *Linear dependency between $\epsilon$ and the input noise in the $\epsilon$-support vector regression*, IEEE Transactions on Neural Networks - ICANN 2001 (2003), 405–410.

[34] A. Lapedes and R. Farber, *How neural nets work*, Neural Information Processing Systems (1987), 442–456.

[35] B. Lillekjendlie, D. Kugiumtzis, and N. Christophersen, *Chaotic time series: Part ii. system identification and prediction*, Modeling, Identification and Control **15** (1994), no. 4, 225–243.

[36] Y. Lin, G. Wahba, H. Zhang, and Y. Lee, *Statistical properties and adaptive tuning of support vector machines*, Machine Learning **48** (2002), 115–136.

[37] E.N. Lorenz, *Deterministic nonperiodic flow*, J. Atmos. Sci. **20** (1963), 130–141.

[38] M.C. Mackey and L. Glass, *Oscillation and chaos in physiological control systems*, Science **197 (4300)** (1977), 287–289.

[39] N. Marono, O. Fontela-Romero, A. Alonso-Betanzos, and B. Guijarro-Berdinas, *Self-organizing maps and functional networks for local dynamic modeling*, ESANN 2003 Proceedings - European Symposium on Artificial Neural Networks, 2003, pp. 39–44.

[40] J. McNames, *Local averaging optimization for chaotic time series prediction*, Neurocomputing **48** (2002), no. 1-4, 279–297.

[41] K. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf, *An introduction to kernel-based learning algorithms*, IEEE Transactions on Neural Networks **12 (2)** (2001), 181–201.

[42] K. Müller, A. Smola, G. Rätsch, B. Schölkopf, O. Kohlmorgen, and V. Vapnik, *Predicting time series with support vector machines*, Artificial Neural Networks - ICANN 97 (W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, eds.), Springer, 1997.

[43] K.A. Oliveira, A. Vannucci, and E.C. Silva, *Using artificial neural networks to forecast chaotic time series*, Physica A **284** (2000), 393–404.

[44] A.E. Omidvar, *Configuring radial basis function network using fractal scaling process with application to chaotic time series prediction*, Chaos, Sol. and Fract. **22** (2004), no. 4, 757–766.

[45] T.S. Parker and L.O. Chua, *Practical numerical algorithms for chaotic systems*, first ed., Springer, New York, 1989.

[46] J. Quiñonero-Candela and L. K. Hansen, *Time series prediction based on the relevance vector machine with adaptive kernels*, International Conference on Acoustics, Speech, and Signal Processing, 2002, pp. 985–988.

[47] L. Ralaivola and F. d'Alche Buc, *Dynamical modeling with kernels for nonlinear time series prediction.*

[48] R. Rosipal, M. Koska, and I. Farkas, *Prediction of chaotic time-series with a resource-allocating RBF network*, Neural Processing Letters **7** (1998), no. 3, 185–197.

[49] D.A. Russell, J.D. Hanson, and E. Ott, *Dimension of strange attractors*, Phys. Rev. Let. **45** (1980), 1175–1178.

[50] T. Sauer, *Time series prediction by using delay coordinate embedding*, Time Series Prediction: Forecasting the Future and Understanding the Past (A.S. Weigend and N.A. Gershenfeld, eds.), Addison-Wesley, 1994, pp. 175–193.

[51] T. Sauer, Y. Yorke, and M. Casdagli, *Embedology*, J. Stat. Phys. **65** (1991), 579–616.

[52] B. Schölkopf, *Statistical learning and kernel methods.*

[53] B. Schölkopf, P. Bartlett, A. Smola, and R. Williamson, *Support vector regression with automatic accuracy control*, Proceedings of ICANN'98: Perspectives in neural computing (Berlin), Springer, 1998, pp. 111–116.

[54] B. Schölkopf, A. Smola, R. Williamson, and P. Bartlett, *New support vector algorithms*, Neural Computation **12 (5)** (2000), 1207–1245.

[55] J. Shawe-Taylor and N. Cristianini, *Robust bounds on generalization from the margin distribution*, Tech. report, Royal Holloway, University of London, 1998.

[56] ———, *On the generalisation of soft margin algorithms*, Tech. report, Royal Holloway, University of London, 2000.

[57] M. Small and C. K. Tse, *Minimum description length neural networks for time series prediction*, Physical Review E (Statistical, Nonlinear, and Soft Matter Physics) **66** (2002), no. 6, 066701.

[58] ———, *Optimal embedding parameters: A modelling paradigm*, Physica D **194** (2004), 283–296.

[59] A. Smola, *Regression estimation with support vector learning machines*, Tech. report, Physik Department, Technische Universität München, 1996.

[60] ———, *Learning with kernels*, Tech. report, GMD Forschungszentrum Informationstechnik, St. Augustin, 1998.

[61] A. Smola and B. Schölkopf, *A tutorial on support vector regression*, Statistics and Computation **13 (3)** (2004), 199–222.

[62] A. Smola, R. Williamson, and B. Scholkopf, *Generalization bounds and learning rates for regularized principal manifolds*, Tech. report, Royal Holloway, University of London, 1998.

[63] F. Takens, *Detecting strange attractors in turbulence*, Dynamical Systems of Turbulence (Berlin) (D. A. Rand and B. S. Young, eds.), vol. 898 of Lecture Notes in Mathematics, Springer, 1981, pp. 366–381.

[64] F. Tay and L. Cao, *Modified support vector machines in financial time series forecasting*, Neurocomputing **48** (2002), no. 1-4, 847–861.

[65] U. Thissen, R. van Brakela, A. P. de Weijerb, W. J. Melssena, and L. M. C. Buyden, *Using support vector machines for time series prediction*, Chemometrics and Intelligent Laboratory Systems **69** (2003), no. 1-2, 35–49.

[66] V. Vapnik, *Estimation of dependencies based on empirical data*, Springer Verlag, New York, 1982.

[67] ———, *The nature of statistical learning theory*, Springer Verlag, New York, 1995.

[68] ———, *Statistical learning theory*, John Wiley & Sons, New York, 1998.

[69] ———, *An overview of statistical learning theory*, IEEE Transactions on Neural Networks **10 (5)** (1999), 998–1000.

[70] V. Vapnik and O. Chapelle, *Bounds on the error expectation for support vector machines*, Neural Computation **12** (2000), 2013–2036.

[71] B.W. Wah and M. Qian, *Violation guided neural-network learning for constrained formulations in time-series predictions*, Int'l Journal on Computational Intelligence and Applications **1** (2001), no. 4, 383–398.

[72] E.A. Wan, *Time series prediction by using a connestionist network with internal delay lines*, Time Series Prediction: Forecasting the Future and Understanding the Past (A.S. Weigend and N.A. Gershenfeld, eds.), Addison-Wesley, 1994, pp. 195–217.

[73] X. Wang, P. Whigham, D. Deng, and M. Purvis, *Time-line hidden markov experts for time series prediction*, Neural Information Processing - Letters and Reviews **3** (2004), no. 2, 39–48.

[74] A.S. Weigend and N.A. Gershenfeld (eds.), *Time series prediction: Forecasting the future and understanding the past*, Addison-Wesley, 1994.

[75] C. Williams, *Prediction with gaussian processes: From linear regression to linear prediction and beyond.*

[76] R. Williamson, A. Smola, and B. Schölkopf, *Generalization performance of regularization networks and support vector machines via entropy numbers of compact operators*, IEEE Transactions on Information Theory **47** (2001), no. 6, 2516–2532.