SPRINGER BRIEFS IN COMPUTER SCIENCE

Tamás Kenesei
János Abonyi

Interpretability of Computational Intelligence-Based Regression Models

Springer

SpringerBriefs in Computer Science

More information about this series at http://www.springer.com/series/10028

Tamás Kenesei · János Abonyi

Interpretability of Computational Intelligence-Based Regression Models

Springer

Tamás Kenesei
Department of Process Engineering
University of Pannonia
Veszprém
Hungary

János Abonyi
Department of Process Engineering
University of Pannonia
Veszprém
Hungary

ISSN 2191-5768 ISSN 2191-5776 (electronic)
SpringerBriefs in Computer Science
ISBN 978-3-319-21941-7 ISBN 978-3-319-21942-4 (eBook)
DOI 10.1007/978-3-319-21942-4

Library of Congress Control Number: 2015947805

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

Preface

Data-driven regression models such as hinging hyperplanes, neural networks and support vector machines are widely applied in control, optimization, and process monitoring. If we had some insight to these black boxes we could have the possibility to validate these models, extract hidden information about relationships among process variables, and support model identification by incorporating prior knowledge.

The key idea of this book is that hinging hyperplanes, neural networks, and support vector machines can be transformed into fuzzy models, and interpretability of the resulting rule-based systems can be ensured by special model reduction and visualization techniques.

The first part of the book deals with the identification of hinging hyperplane-based regression trees. The operating regime of the model is recursively partitioned by a novel fuzzy c-regression clustering-based technique. The resultant compact regression tree consists of local linear models whose model structure is favored in model-based control solutions, as in model predictive control.

The next section deals with the validation, visualization and structural reduction of neural networks based on the transformation of the hidden layer of the network into an additive fuzzy rule-based system.

Finally, based on the analogy of support vector regression and fuzzy models, a three-step model reduction algorithm will be proposed to get interpretable fuzzy regression models on the basis of support vector regression.

Real-life utilization of the developed algorithms is shown by section-wise examples taken from the area of process engineering. The discussion of the proposed algorithms is supported by over 35 figures; more than 90 references that give a good overview of the current state of nonlinear regression, and suggested further reading material for students and researchers interested in the details; algorithms that aim to understand the methods in detail and help implement them; and over ten examples with Matlab files downloadable from the authors' website (www.abonyilab.com).

The research of János Abonyi was realized in the frames of TÁMOP 4.2.4. A/2-11-1-2012-0001, “National Excellence Program elaborating and operating an inland student and researcher personal support system”.

Veszprém Tamás Kenesei
December 2014 János Abonyi

Contents

Symbols and Acronyms

Symbols

α	Filter parameter
β	Firing strength
θ	Model parameter
ρ	Conditionality
δ	Rule consequent
Λ	A priori constraints
λ	Weighting coefficient
ε	Error tolerance
$\mu_{i,k}$	Membership value, where i stands for the clusters and k for the data points ($k = 1...n$)
τ	Bias
ϕ	Feature mapping
C	Cost coefficient
E, J	Cost function
n_a	Output order
H_p	Prediction horizon
H_c	Control horizon
n_b	Input order
N_i	Input space dimension
N_x	Number of support vectors
N_R	Reduced set expansion
$Q(u)$	Cartridge heater performance
Q_M	Maximum power
u	Heating signal
T_{out}	Outlet temperature
S	Similarity measure
k	Kernel function
$\mathbf{U}$	Fuzzy partition matrix
$\mathbf{v}$	Cluster center

x	Independent variable
ξ_i, ξ_i^*	Slack variables
y	Dependent variable
$*$	Interactive or operator

Acronyms

ANN	Artificial Neural Network
APC	Advanced process control
CSTR	Continuous stirred tank reactor
DT	Decision tree
FAS	Fuzzy additive system
FCRM	Fuzzy c-regression clustering method
FIS	Fuzzy system
HH	Hinging hyperplanes
KDD	Knowledge data discovery in databases
LOLIMOT	Locally linear model tree
MDS	Multi-dimensional scaling
MLP	Multilayer perceptron
MPC	Model predictive control
NARX	Nonlinear autoregressive with exogenous input
NN	Neural network
OLS	Orthogonal least squares
PDFS	Positive definite fuzzy system
RBF	Radial basis function
RMSE	Root mean square error
RS	Reduced set method
SVM	Support vector machine
SVR	Support vector regression
TS	Takagi-Sugeno fuzzy model

Chapter 1
Introduction

With a combination of computational intelligence tools such as hinging hyperplanes (HHs), support vector regression (SVR), artificial neural networks (ANNs) and fuzzy models, powerful and interpretable models can be developed. This introduction presents the motivation for handling these techniques in one integrated framework and describes the structure of the book.

1.1 Data-Driven Modeling

Information for modeling and systems identification can be obtained from different sources. According to the type of available information, three basic levels of modeling are defined:

White-box or first-principles modeling A complete mechanistic model is constructed from a priori knowledge and physical insight. Here dynamic models are derived based on mass, energy and momentum balances of the process [1].

Fuzzy logic modeling A linguistically interpretable rule-based model is formed based on the available expert knowledge and measured data [1].

Black-box model or empirical model No physical insight is available or used, but the chosen model structure belongs to families that are known to have good flexibility and have been "successful in the past". Model parameters are identified based on measurement data [1].

This means, if we have good mechanistic knowledge about the system, this can be transformed into a white-box model described by analytical (differential) equations. If we have information like human experience described by linguistic rules and variables, the mechanistic modeling approach is useless and the application of rule-based approaches like fuzzy logic is more appropriate [2, 3]. Finally there may be situations where the most valuable information comes from input-output data collected during operation. In this case, the application of black-box models is the

T. Kenesei and J. Abonyi, *Interpretability of Computational Intelligence-Based Regression Models*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-21942-4_1

best choice. These black-box models are especially valuable when an accurate model of the process dynamics is needed. Therefore, nonlinear black-box modeling is a challenging and promising research field [4–8].

Black-box models are especially valuable when an accurate model of the process dynamics is needed. In order to perform a successful data-driven model the following steps have to be carried out [1]:

1. Selection of model structure and complexity
2. Design of excitation signals used or identification
3. Identification of model parameters
4. Model validation

Data analysts are often distrustful of the application of nonlinear black-box models since they are not interpretable. If we would have some insight into these black boxes we could validate these models, extract hidden information about the relationships among the variables, select the model structure based on this knowledge, and support model identification by incorporating some prior knowledge.

For these purposes novel model identification methods and interpretable, robust and transparent models are needed. Since we are interested in extraction of knowledge from process data, tools and methodologies of data mining should also be efficiently utilized. Historically the notion of finding useful patterns in data has been given a variety of names, including data mining, knowledge extraction, information discovery, and data pattern processing. The term data mining has been mostly used by statisticians, data analysts, and the management information systems (MIS) communities [9]. Data mining is not just a simple tool, but a complex process consisting of multiple steps; hence this process must be integrated into the supported activity. The process of data-based knowledge discovery can be seen in Fig. 1.1. Introducing the knowledge discovery process allows us to give a brief definition of data mining: *Data mining is a decision support process to give valid, useful and a priori unknown reliable information from data sources* [10].

To understand this definition the following keywords must be further investigated [1]:

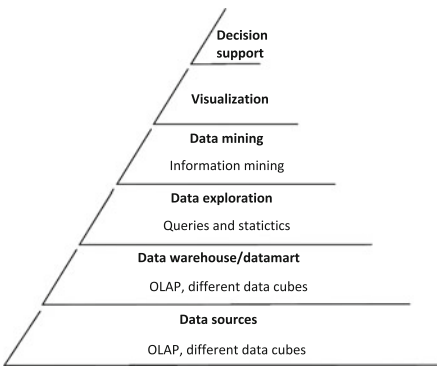

Fig. 1.1 Steps of the knowledge discovery process

Process Data mining is not a product-ready delivered software generating automatically consumable knowledge from stored data, but it is a complex process consisting of well-defined steps. Regression technique take place during model preparation. Improved model identification algorithms and interpretable models ensure the quality of the acquired information at the end of the KDD process.

Valid Mined information must be accurate and statistically significant. Validity states not only accuracy but also completeness.

Useful It is not enough to generate valid knowledge with the help of data mining; explored knowledge must be utilizable for the problem exactly defined. Unfortunately measuring usefulness is not always solved, as sometimes the effect of the used information cannot be measured with monetary tools.

Preliminarily not known Strictly speaking, data-based knowledge discovery has two aims: confirmation and discovery. Confirmation means strengthening the hypothesis of the data expert while discovery stands for identification of patterns generated by the examined system. The aim of data mining basically is to identify discoverable knowledge to define predictive or descriptive functions. Regression-based methods are predictive exercises defining future, unknown properties or behaviors of the modeled process.

Exact The result of data mining must be easily interpretable, and the model should not deviate from reality.

The key challenge of data mining is to capture potential information from opaque data sources and transform data to a more compact, abstract, informative and easy-to-use format. Hence, data mining looks for trends and patterns in large databases. Knowledge delivered by data mining exists in the form of an interpretable model or information represented in decision trees, rule bases, networks or mathematical equations.

The aim of this book is to extract interpretable regression models to foster the applicability of these models.

1.2 Interpretability and Model Structure Identification

Zadeh stated in Principle of Incompatibility [11] that "*as the complexity of a system increases our ability to make precise and yet significant statements about its behavior diminishes until a threshold is reached beyond which precision and significance (or relevance) become almost mutually exclusive characteristics.*"

Obtaining a high degree of interpretability with sizeable complexity is a contradictory purpose and in practice one of these properties prevails over the other (see Fig. 1.2). The problem becomes much more difficult to handle when model reliability and maintenance are included in the conditions.

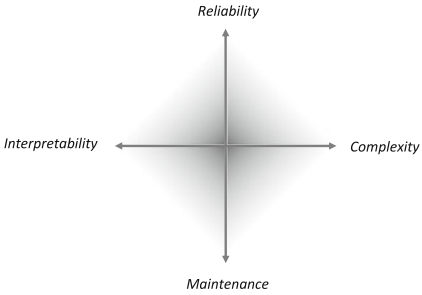

Fig. 1.2 Trade-offs in modeling

In most studies of systems identification it is assumed that there is an optimal functional structure describing the relationship between measured input and output data. It is very difficult to find a functional structure for a nonlinear system. Generally speaking, the structure identification of a system has to solve two problems: one is to find input variables and one is to find input-output relations. For static systems statistical techniques, correlation analysis and modeling performance-based methods proposed by Sugeno [12–14] and Jang [15] can be used. For dynamical systems, the selection of the relevant model inputs is identical to the determination of the model order of the NARX model.

The identified model must be validated as well. If the model is validated by the same data set from which it was estimated, the accuracy of the model always improves as the complexity of the model structure increases. In practice, a trade-off is usually sought between model accuracy and complexity, and there are several approaches to compensate for this automatic increase of the modeling performance.

Figure 1.3 shows the connection between modeling error and model complexity [1]. As this figure shows, selection of the proper model structure is a complex task that requires careful selection of training and validation datasets, proper cost functions and proper optimization strategies or heuristics that support the modeler.

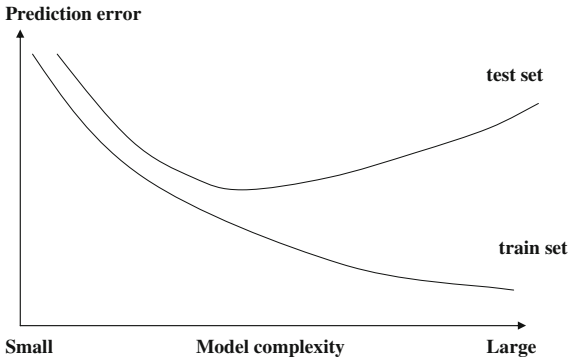

Fig. 1.3 Model complexity versus modeling error

1.3 Computational Intelligence-Based Models

Most problems arising in engineering practice require data-driven modeling of nonlinear relationships between experimental and technological variables. The complexity of nonlinear regression techniques is gradually expanding with the development of analytical and experimental techniques, hence model structure and parameter identification is a current and important topic in the field of nonlinear regression, from not just the scientific but from the industrial points of view as well.

In line with these expectations, and taking the interpretability of regression models as a basic requirement, the aim of this book is the development of robust computational intelligence models in order to solve nonlinear regression identification tasks.

Tools from the armory of computational intelligence (also referred to as soft computing) have been the focus of research recently, since soft computing techniques are used for fault detection, forecasting of time-series data, inference, hypothesis testing, and modeling of causal relationships (regression techniques) in process engineering.

The meaning of soft computing was originally tailored in the early 1990s by Zadeh [16]. Soft computing refers to a collection of computational techniques in computer science, artificial intelligence, machine learning and some engineering disciplines, to solve two cardinal problems:

- learning from experimental data (examples, samples, measurements, records, patterns) by neural networks and support vector-based techniques;
- embedding existing structured human knowledge (experience, expertise, heuristic) into fuzzy models [17]

These approaches attempt to study, model, and analyze very complex phenomena: those for which more conventional methods have not yielded low-cost, analytic, and complete solutions. Earlier computational approaches (hard computing) could model and precisely analyze only relatively simple systems.

More complex systems arising in biology, medicine, the humanities, management sciences, and similar fields often remained intractable to conventional mathematical and analytical methods. Where hard computing schemes, striving for exactness and full truth, fail to render the given problem, soft computing techniques deliver robust, efficient and optimal solutions to capture available design knowledge for further analysis.

Generally speaking, soft computing techniques resemble biological processes more closely than traditional techniques, which are largely based on formal logical systems, such as sentential logic and predicate logic, or rely heavily on computer-aided numerical analysis. Hence, in real life, a high degree of uncertainty should be taken into account during the identification process. Soft computing tries to solve this challenge by exploiting tolerance for imprecision, uncertainty and partial truth in order to attain robustness and transparency at low cost.

Many systems are not amenable to conventional modeling approaches because of the lack of precise, formal knowledge about the system, due to strongly nonlinear behavior, high degree of uncertainty, or time-varying characteristics. Computational

intelligence, the technical umbrella of hinging hyperplanes (HH) [18–20], support vector regression (SVR) [21–25], artificial neural networks (ANNs) [26–31] and fuzzy logic [2, 3, 32], has been recognized as a powerful tool which is tolerant to imprecision and uncertainty, and can facilitate the effective development of models by combining information from different sources, such as first-principles models, heuristics and data.

Among the techniques of computational intelligence, ANNs attempt to mimic the structures and processes of biological neural systems. They provide powerful analysis properties such as complex processing of large input/output information arrays, representing complicated nonlinear associations among data. Support vector regression in its nature is very similar to ANNs; on the other hand, HH models can be a good alternative to NNs.

Based on [1], Table 1.1 summarizes several critera that can be used to evaluate these modeling techniques. The studied nonlinear regression techniques have usually robust modeling structure; however the resulting model is often a non-interpretable black-box model. This book focuses on the identification, utilization and interpretability of ANNs, HHs and SVR in the realm of modeling, identification and control of nonlinear processes.

Table 1.1 Evaluation criteria

Property	Description
Interpolation behavior	Character of the model output between training data samples
Extrapolation behavior	Character of the model outside region of training data
Locality	Locality, globality of the basis functions
Accuracy	Model accuracy with given number of parameters
Smoothness	Smoothness of model output
Sensitivity to noise	Effect of noise on model behavior
Parameter optimization	Can linear and nonlinear model parameters be estimated?
Structure optimization	Possibilities of model structure and complexity optimization
Online adaptation	Possibilities of online adaptable model
Training speed	Speed of model parameter estimation
Evaluation speed	Speed of model evaluation
Curse of dimensionality	Model scale-up to higher input space dimensions
Interpretation	Interpretation of model parameters and model structure
Incorporation of constraints	Difficulty of constraint incorporation
Usage	Acceptability and penetration of modeling structure

1.4 Motivation and Outline of the Book

This book has two aims: to introduce new algorithms for nonlinear regression (hinging hyperplanes) and to highlight ways to transform black-box nonlinear regression models (neural networks and support vector regression) to a transparent and interpretable fuzzy rule base (see Fig. 1.4). These techniques together form a framework to utilize combination-of-tools methods to understand, visualize and validate nonlinear black-box models.

Three algorithms were examined based on the evaluation criteria system discussed in Sect. 1.2 in detail, namely, identification of regression tree-based hinging hyperplane, neural networks, and support vector regression. Application of these techniques eventuate in black-box models at the first step. It will be shown how interpretability could be maintained during model identification with utilization of applicable visualization and model structure reduction techniques within the fuzzy modeling framework.

Chapter 2 (**Hinging Hyperplanes**) deals with the identification of hinging hyperplane-based regression trees. The results of the developed algorithm prove that the implementation of a priori constraints enables the fuzzy c-regression clustering technique to identify hinging hyperplane models. Application of this technique recursively on the partitioned input space ends up in a regression tree capable of modeling and even implementing model predictive control of technological data coming from real-life applications. According to the evaluation system mentioned in Sect. 1.2 the development algorithm contains major developments in hinge model identification; since the proposed method has higher *accuracy*, the tree-based representation enables better *structure* and *parameter optimization* and helps *interpretation* of model parameters and model structure.

Chapter 3 (**Visualization and Reduction of Neural Networks**) deals with the validation, visualization and structural reduction of neural networks in order to achieve better *interpretability*. By applying orthogonal least squares and similarity measure-based techniques, *structure optimization* is also performed. It is described in detail that the hidden layer of the neural network can be transformed to an additive fuzzy rule base.

Chapter 4 (**Interpretable Support Vector Regression**) describes connections between fuzzy regression and support vector regression, and introduces a three-

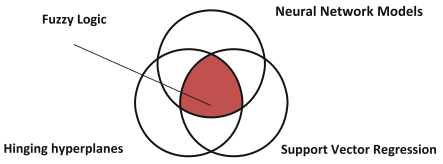

Fig. 1.4 Framework of the book

step reduction algorithm to get interpretable fuzzy regression models on the basis of support vector regression. This combination-of-tools technique retains good *generalization* behavior and noise *insensitivity* of the support vector regression; however, it keeps the identified model *interpretable*, and with the application of reduction techniques *structure optimization* is also achieved.

Real-life utilization of the developed algorithms is shown by sectionwise examples taken from the area of chemical engineering. Finally, Chap. 5 summarizes the new scientific results.

The proposed framework supports the structure and parameter identification of regression models. To achieve this goal, structure optimization techniques, like orthogonal least squares (OLS) and decision tree-based model representations, are applied. OLS is described in detail in Appendix B.

Source code of the utilized software written in Matlab can be found at www.abonyilab.com website.

Chapter 2
Interpretability of Hinging Hyperplanes

The hinging hyperplane model was proposed by Breiman [20]. This type of nonlinear model is often referenced in the literature since it suffers from convergency and range problems [19, 33–35]. Methods such as a penalty of the hinging angle were proposed to improve Breiman's algorithm [18]; alternatively, the Gauss-Newton algorithm can be used to obtain the final nonlinear model [34]. Several application examples have also been published in the literature; e.g., it can be used in the identification of piecewise affine systems via mixed-integer programming [36], and this model also lends itself to forming hierarchical models [19].

In this chapter a much more applicable algorithm is proposed for hinging hyperplane identification. The key idea is that in a special case ($c = 2$), the fuzzy c-regression method (FCRM) [37] can be used for identifying hinging hyperplane models. To ensure that two local linear models used by the fuzzy c-regression algorithm form a hinging hyperplane function, it has to be guaranteed that local models intersect each other in the operating regime of the model. The proposed constrained FCRM algorithm is able to identify one hinging hyperplane model; therefore, to generate more complex regression trees, the described method should be recursively applied. Hinging hyperplane models containing two linear submodels divide the operating region of the model into two parts, since hinging hyperplane functions define a linear separating function in the input space of the hinging hyperplane function. These separations result in a regression tree where branches correspond to linear divisions of the operating regime based on the hinge of the hyperplanes at a given node. This type of partitioning can be considered as the crisp version of a fuzzy regression-based tree described in [38]. Fortunately, in the case of a hinging hyperplane-based regression tree there is no need to select the best splitting variable at a given node, but, on the other hand, it is not as interpretable as regression trees utilizing univariate decisions at nodes.

To support the analysis and building of this special model structure, novel model performance and complexity measures are presented in this work. Special attention is given to modeling and controlling nonlinear dynamical systems. Therefore, an application example related to the Box-Jenkins gas furnace benchmark identification

T. Kenesei and J. Abonyi, *Interpretability of Computational Intelligence-Based Regression Models*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-21942-4_2

problem is added. It will also be shown, that thanks to the piecewise linear model structure, the resulting regression tree can be easily utilized in model predictive control. A detailed application example related to the model predictive control of a water heater will demonstrate the benefits of the proposed framework.

A critical step in the application of model-based control is the development of a suitable model for the process dynamics. This difficulty stems from lack of knowledge or understanding of the process to be controlled. Fuzzy modeling has been proven to be effective for the approximation of uncertain nonlinear processes. Recently, nonlinear black-box techniques using fuzzy and neuro-fuzzy modeling have received a great deal of attention [39]. Readers interested in industrial applications can find an excellent overview in [40]. Details of relevant model-based control applications are well presented in [41, 42].

Most nonlinear identification methods are based on the NARX (Nonlinear AutoRegressive with eXogenous input) model [8]. The use of NARX black-box models for high-order dynamic processes in some cases are impractical. Data-driven identification techniques alone may yield unrealistic NARX models in terms of steady-state characteristics, local behavior and unreliable parameter values. Moreover, the identified model can exhibit regimes which are not found in the original system [42]. This is typically due to insufficient information content of the identification data and the overparametrization of the model. This problem can be remedied by incorporating prior knowledge into the identification method by constraining the parameters of the model [43]. Another way to reduce the effects of overparametrization is to restrict the structure of the NARX model, using, for instance, the Nonlinear Additive AutoRegressive with eXogenous input (NAARX) model [44]. In this book a different approach is proposed; a hierarchical set of local linear models are identified to handle complex systems dynamics.

Operating regime-based modeling is a widely applied technique for identification of these nonlinear systems. There are two approaches building operating regime-based models. An additive model uses the sum of certain basis functions to represent a non-linear system, while partitioning approach partitions the input space recursively to increase modeling accuracy locally [18]. Models generated by this approach are often represented by trees [45]. Piecewise linear systems [46] can be easily represented in a regression tree structure [47]. A special type of regression tree is called the locally linear model tree, which combines a heuristic strategy for input space decomposition with a local linear least squares optimization (like LOLIMOT [1]). These models are hierarchical models consisting of nodes and branches. Internal nodes represent tests on input variables of the model, and branches correspond to outcomes of the tests. Leaf (terminal) nodes contains regression models in the case of regression trees.

Thanks to the structured representation of the local linear models, hinging hyperplanes lend themselves to a straightforward incorporation into model-based control schemes. In this chapter this beneficial property is demonstrated in the design of an instantaneous linearization-based model predictive control algorithm [32].

This chapter organized as follows: the next section discusses how hinging hyperplane function approximation is done with the FCRM identification approach. The

description of the tree growing algorithm and the measures proposed to support model building are given in Sect. 2.2. In Sect. 2.3, application examples are presented, while Sect. 2.4 concludes the chapter.

2.1 Identification of Hinging Hyperplanes

2.1.1 Hinging Hyperplanes

The following section gives a brief description of the hinging hyperplane approach on the basis of [18, 34, 48], followed by a description of how the constraints can be incorporated into FCRM clustering.

For a sufficiently smooth function $f(\mathbf{x}_k)$, which can be linear or nonlinear, assume that regression data $\{\mathbf{x}_k, y_k\}$ is available for $k = 1, \ldots, N$. Function $f(\mathbf{x}_k)$ can be represented as the sum of a series of hinging hyperplane functions $h_i(\mathbf{x}_k)\, i = 1, 2, \ldots, K$. Breiman [20] proved that we can use hinging hyperplanes to approximate continuous functions on compact sets, guaranteeing a bounded approximation error

$$\|e_n\| = \|f - \sum_{i=1}^{K} h_i(\mathbf{x})\| \leq (2R)^4\, c^2/K, \tag{2.1}$$

where K is the number of hinging hyperplane functions, R is the radius of the sphere in which the compact set is contained, and c is such that

$$\int \|w\|^2 |f\,(w)\,| dw = c < \infty. \tag{2.2}$$

The approximation with hinging hyperplane functions can get arbitrarily close if a sufficiently large number of hinging hyperplane functions are used. The sum of the hinging hyperplane functions, $\sum_{i=1}^{K} h_i(\mathbf{x}_k)$, constitutes a continuous piecewise linear function. The number of input variables n in each hinging hyperplane function and the number of hinging hyperplane functions K are two variables to be determined. The explicit form for representing a function $f(\mathbf{x}_k)$ with hinging hyperplane functions becomes (see Fig. 2.1)

$$f(\mathbf{x}_k) = \sum_{i=1}^{K} h_i(\mathbf{x}_k) = \sum_{i=1}^{K} \langle \max \mid \min \rangle \left(\mathbf{x}_k^T \theta_{1,i}, \mathbf{x}_k^T \theta_{2,i} \right), \tag{2.3}$$

where $\langle \max \mid \min \rangle$ means max or min.

Suppose two hyperplanes are given by:

$$y_k = \mathbf{x}_k^T \theta_1,\, y_k = \mathbf{x}_k^T \theta_2, \tag{2.4}$$

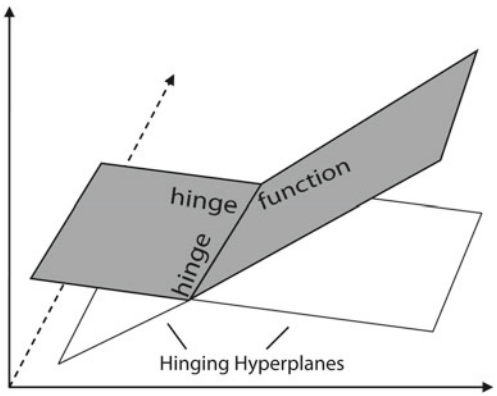

Fig. 2.1 Basic hinging hyperplane definitions

where $\mathbf{x}_k = \left[x_{k,0}, x_{k,1}, x_{k,2}, \ldots, x_{k,n}\right]$, $x_{k,0} \equiv 1$, is the kth regressor vector and y_k is the kth output variable. These two hyperplanes are continuously joined together at $\{\mathbf{x} : \mathbf{x}^T (\theta_1 - \theta_2) = 0\}$, as can be seen in Fig. 2.1. As a result they are called *hinging hyperplanes*. The joint $\triangle = \theta_1 - \theta_2$, is defined as *hinge* for the two hyperplanes $y_k = \mathbf{x}_k^T \theta_1$ and $y_k = \mathbf{x}_k^T \theta_2$. The solid/shaded parts of the two hyperplanes are explicitly given by

$$y_k = \max(\mathbf{x}_k^T \theta_1, \mathbf{x}_k^T \theta_2) \text{ or } y_k = \min(\mathbf{x}_k^T \theta_1, \mathbf{x}_k^T \theta_2). \tag{2.5}$$

The hinging hyperplane method has some interesting advantages for nonlinear function approximation and identification:

1. Hinging hyperplane functions could be located by a simple computationally efficient method. In fact, hinging hyperplane models are piecewise linear models; the linear models are usually obtained by repeated use of the linear least squares method, which is very efficient. The aim is to improve the whole identification method with more sophisticated ideas.
2. For nonlinear functions that resemble hinging hyperplane functions, the hinging hyperplane method has very good and fast convergence properties.

The hinging hyperplane method practically combines some advantages of neural networks (in particular, the ability to handle very large dimensional inputs) and constructive wavelet-based estimators (availability of very fast training algorithms).

The essential hinging hyperplane search problem can be viewed as an extension of the linear least squares regression problem. Linear least squares regression aims to find the best parameter vector $\widehat{\theta}$ by minimizing a quadratic cost function with the regression model that gives the best linear approximation to y. For nonsingular data matrix $\mathbf{X}$, the linear least squares estimate $y = \mathbf{x}^T \theta$ is always uniquely available. The hinging hyperplane search problem, on the other hand, aims to find the two parameter vectors θ_1 and θ_2, defined by

$$[\theta_1, \theta_2] = \arg\min_{\theta_1,\theta_2} \sum_{k=1}^{N} \left[\langle \max \mid \min \rangle \left(y_k - \mathbf{x}_k^T \theta_1,\, y_k - \mathbf{x}_k^T \theta_2 \right) \right]^2 . \tag{2.6}$$

A brute force application of the Gauss-Newton method can solve the above described optimization problem. However, two problems exist [18]:

1. High computational requirement. The Gauss-Newton method is computationally intensive. In addition, since the cost function is not continuously differentiable, the gradients required by the Gauss-Newton method cannot be given analytically. Numerical evaluation is thus needed, which has high computational requirement.
2. Local minima. There is no guarantee that the global minimum can be obtained. Therefore, an appropriate initial condition is crucial.

2.1.2 Improvements in Hinging Hyperplane Identification

The proposed identification algorithm applies a much simpler optimization method, the so-called alternating optimization, which is a heuristic optimization technique and has been applied for several decades for many purposes; therefore, it is an exhaustively tested method in nonlinear parameter and structure identification as well. Within the hinging hyperplane function approximation approach, the two linear submodels can be identified by the weighted linear least squares approach, but their operating regimes (where they are valid) are still an open question.

For that purpose, the fuzzy c-regression model (further referred as FCRM and proposed by Hathaway and Bezdek [37]) was used. This technique is able to partition the data and determine the parameters of the linear submodels simultaneously. With the application of alternating optimization techniques and by taking advantage of the linearity in $(y_k - \mathbf{x}_k^T \theta_1)$ and $(y_k - \mathbf{x}_k^T \theta_2)$, an effective approach is given for hinging hyperplane function identification; hence the FCRM method for a special case ($c = 2$) is able to identify hinging hyperplanes. The proposed procedure is attractive from the local minima point of view as well, because in this way, although the problem is not avoided, it is transformed into a deeply discussed problem, namely the cluster validity problem.

The following quadratic cost function can be applied for the FCRM method:

$$E_m(\mathbf{U}, \{\theta_i\}) = \sum_{i=1}^{c} \sum_{k=1}^{N} (\mu_{i,k})^m E_{i,k}(\theta_i) \,, \tag{2.7}$$

where $m \in \langle 1, \infty)$ denotes a weighting exponent which determines the fuzziness of the resulting clusters, while θ_i represents the parameters of local models and $\mu_{i,k} \in \mathbf{U}$ is the membership degree, which could be interpreted as a weight representing the extent to which the value predicted by the model $f_i(\mathbf{x}_k, \theta_i)$ matches y_k. The prediction error is defined by

$$E_{i,k} = \left(y_k - f_i(\mathbf{x}_k; \theta_i)\right)^2, \tag{2.8}$$

but other measures can be applied as well, provided they fulfill the minimizer property stated by Hathaway and Bezdek [37].

One possible approach to the minimization of the objective function (2.7) is the group coordinate minimization method that results in the following algorithm:

- **Initialization** Given a set of data $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, specify c, the structure of the regression models (2.8) and the error measure (2.7). Choose a weighting exponent $m > 1$ and a termination tolerance $\varepsilon > 0$. Initialize the partition matrix randomly.
- **Repeat** For $l = 1, 2, \ldots$
- Step 1 Calculate values for the model parameters θ_i that minimize the cost function $E_m(\mathbf{U}, \{\theta_i\})$.
- Step 2 Update the partition matrix

$$\mu_{i,k}^{(l)} = \frac{1}{\sum_{j=1}^{c} (E_{i,k}/E_{j,k})^{2/(m-1)}}, \ 1 \le i \le c, \ 1 \le k \le N \tag{2.9}$$

until $||\mathbf{U}^{(l)} - \mathbf{U}^{(l-1)}|| < \varepsilon$.

A specific situation arises when the regression functions f_i are linear in the parameters θ_i, $f_i(\mathbf{x}_k; \theta_i) = \mathbf{x}_{i,k}^T \theta_i$, where $\mathbf{x}_{i,k}$ is a known arbitrary function of $\mathbf{x}_k$. In this case, the parameters can be obtained as a solution of a set of the weighted least squares problem where the membership degrees of the fuzzy partition matrix $\mathbf{U}$ serve as the weights.

The N data pairs and the membership degrees are arranged in the following matrices:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_{i,1}^T \\ \mathbf{x}_{i,2}^T \\ \vdots \\ \mathbf{x}_{i,N}^T \end{bmatrix}, \ \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}, \ \Phi_i = \begin{bmatrix} \mu_{i,1} & 0 & \cdots & 0 \\ 0 & \mu_{i,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mu_{i,N} \end{bmatrix}. \tag{2.10}$$

The optimal parameters θ_i are then computed by

$$\theta_i = [\mathbf{X}^T \Phi_i \mathbf{X}]^{-1} \mathbf{X}^T \Phi_i \mathbf{y}. \tag{2.11}$$

Applying $c = 2$ during FCRM identification, these models can be used as base identifiers for hinging hyperplane functions. For hinging hyperplane function identification purposes, two prototypes have to be used by FCRM ($c = 2$), and these prototypes must be linear regression models. However, these linear submodels have to intersect each other within the operating regime covered by the known data points (within the hypercube expanded by the data). This is a crucial problem in the hinging hyperplane identification area [18]. To take into account this point of view as well,

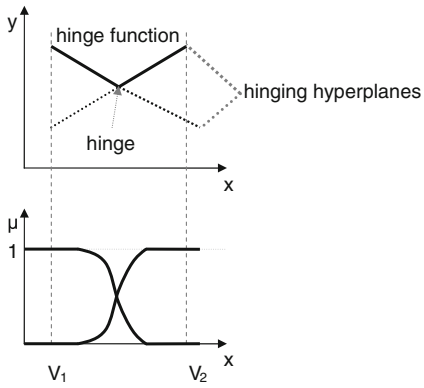

Fig. 2.2 Hinging hyperplane identification restrictions

constraints have to be taken into consideration as follows. Cluster centers $\mathbf{v}_i$ can also be computed from the result of FCRM as the weighted average of the known input data points:

$$\mathbf{v}_i = \frac{\sum_{k=1}^{N} \mathbf{x}_k \mu_{i,k}}{\sum_{k=1}^{N} \mu_{i,k}}, \tag{2.12}$$

where the membership degree $\mu_{i,k}$ is interpreted as a weight representing the extent to which the value predicted by the model matches y_k. These cluster centers are located in the 'middle' of the operating regime of the two linear submodels. Because the two hyperplanes must cross each other, the following criteria can be specified (see Fig. 2.2):

$$\begin{aligned} &\mathbf{v}_1(\theta_1 - \theta_2) < 0 \text{ and } \mathbf{v}_2(\theta_1 - \theta_2) > 0 \text{ or} \\ &\mathbf{v}_1(\theta_1 - \theta_2) > 0 \text{ and } \mathbf{v}_2(\theta_1 - \theta_2) < 0. \end{aligned} \tag{2.13}$$

These relative constraints can be used to take into account the constraints above:

$$\lambda_{rel,1,2} \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \leq 0 \text{ where } \lambda_{rel,1,2} = \begin{bmatrix} \mathbf{v}_1 & -\mathbf{v}_1 \\ -\mathbf{v}_2 & \mathbf{v}_2 \end{bmatrix}. \tag{2.14}$$

When linear equality and inequality constraints are defined on these prototypes, quadratic programming (QP) has to be used instead of the least squares method. This optimization problem still can be solved effectively compared to other constrained nonlinear optimization algorithms.

Local linear constraints applied to fuzzy models can be grouped into the following categories according to their validity region:

- **Local constraints** are valid only for the parameters of a regression model, $\lambda_i \theta_i \leq \omega_i$.
- **Global constraints** are related to all of the regression models, $\lambda_{gl} \theta_i \leq \omega_{gl}$, $i = 1, \ldots, c$.

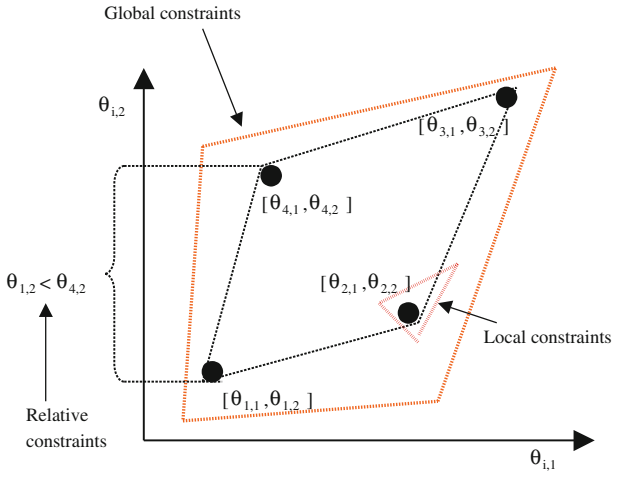

Fig. 2.3 Hinging hyperplane model with four local constraints and two parameters

- **Relative constraints** define the relative magnitude of the parameters of two or more regression models:

$$\lambda_{rel,i,j} \begin{bmatrix} \theta_i \\ \theta_j \end{bmatrix} \leq \omega_{rel,i,j}. \tag{2.15}$$

An example of these types of constraints is illustrated in Fig. 2.3.

In order to handle relative constraints, the set of weighted optimization problems has to be solved simultaneously. Hence, the constrained optimization problem is formulated as follows:

$$\min_{\theta} \left\{ \frac{1}{2} \theta^T \mathbf{H} \theta + \mathbf{c}^T \theta \right\}, \tag{2.16}$$

with $\mathbf{H} = 2\mathbf{X}'^T \Phi \mathbf{X}'$, $\mathbf{c} = -2\mathbf{X}'^T \Phi \mathbf{y}'$, where

$$\mathbf{y}' = \begin{bmatrix} \mathbf{y} \\ \mathbf{y} \\ \vdots \\ \mathbf{y} \end{bmatrix}, \quad \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_c \end{bmatrix}, \tag{2.17}$$

$$\mathbf{X}' = \begin{bmatrix} \mathbf{X}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{X}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{X}_c \end{bmatrix}, \quad \Phi = \begin{bmatrix} \Phi_1 & 0 & \cdots & 0 \\ 0 & \Phi_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Phi_c \end{bmatrix}, \tag{2.18}$$

where Φ_i contains local membership values and the constraints on θ:

$$\Lambda\theta \leq \omega, \tag{2.19}$$

with

$$\lambda = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_c \\ \lambda_{gl} & 0 & \cdots & 0 \\ 0 & \lambda_{gl} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{gl} \\ \{\lambda_{rel}\} & & & \end{bmatrix}, \ \omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_c \\ \omega_{gl} \\ \omega_{gl} \\ \vdots \\ \omega_{gl} \\ \{\omega_{rel}\} \end{bmatrix}. \tag{2.20}$$

Referring back to Fig. 2.1, it can be concluded that with this method both parts of the intersected hyperplanes are described and the part ($\langle max|min\rangle$) that describes the training data in the most accurate way is selected.

2.2 Hinging Hyperplane-Based Binary Trees

So far, the hinging hyperplane function identification method has been presented. The proposed technique can be used to determine the parameters of one hinging hyperplane function. The classical hinging hyperplane approach can be interpreted by identifying K hinging hyperplane models consisting of global model pairs, since these operating regimes cover the whole N dataset. This representation leads to several problems during model identification and also renders model interpretability more difficult. To overcome this problem, a tree structure is proposed where the data is recursively partitioned into subsets, while each subset is used to form models of lower levels of the tree. The concept is illustrated in Fig. 2.4, where the membership functions and the identified hinging hyperplane models are also shown.

During the identification the following phenomena can be taken into consideration (and can be considered as benefits too):

- When using the hinging hyperplane function there is no need to find splitting variables at the nonterminal nodes, since this procedure is based on the hinge.
- A populated tree is always a binary tree, either balanced or non-balanced, depending on the algorithm (greedy or non-greedy). It is based on a binary tree; the hinge splitting the $\mathbf{x}$ data pertains to the left side of the hinge and θ_1 always goes to the left child; and the right side behaves the similarly. For example, given a simple symmetrical binary tree structure model, the first level contains one hinging

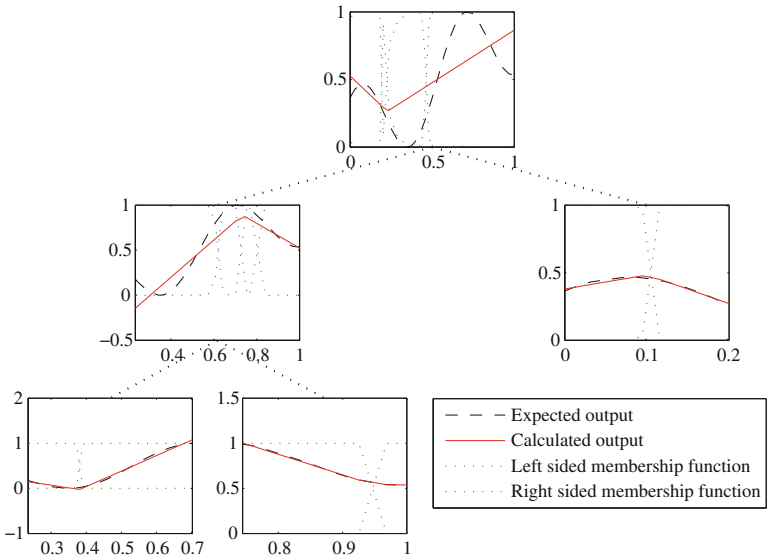

Fig. 2.4 Hinging hyperplane-based regression tree for basic data sample in case of greedy algorithm

hyperplane function, the second level contains two hinging hyperplane functions, the third level contains four hinges, and in general the kth level contains $2^{(k-1)}$ hinging hyperplane functions.

Concluding the above and obtaining the parameters θ during recursive identification, the following cost function has to be minimized:

$$E(\{\theta_i\}, \pi) = \sum_{i=1}^{K} \pi_i E_{m_i}(\theta_i), \tag{2.21}$$

where K is the number of the hinge functions (nodes), and π is the binary ($\pi_i \in 0, 1$) terminal set, indicating that the given node is a final linear model ($\pi_i = 1$), and can be incorporated as a terminal node of the identified piecewise model.

A growing algorithm can be either balanced or greedy. In the balanced case the identification algorithm builds the tree till the desired stopping criteria, while the greedy one continues the tree building by choosing a node for splitting that performs worst during the building procedure. Hence, this operating regime needs more local models for better model performance. For a greedy algorithm, the crucial item is the selection of a good stopping criterion. Any of the following can be used to determine whether to continue the tree growing process or stop the procedure:

1. The loss function becomes zero. This corresponds to the situation where the size of the data set is less than or equal to the dimension of the hinge. Since the hinging hyperplanes are located by linear least squares, when the number of data points is

equal to the number of parameters to be determined, the result is exact, provided the matrix is not singular.

2. $E = E_1 + E_2$, where $E_i = \sum_{k=1}^{N} \mu_{i,k} \left(y_k - f_i(\mathbf{x}_k; \theta_i)\right)^2$ represents the performances of the left-and right-hand side models of the hinge. During the growth of the binary tree, the loss function is always nonincreasing, so E should be always smaller than the performance of the parent node. When no decrease is observed in the loss function, the growing should be stopped.
3. The tree-building process reaches the predefined tree depth.
4. All of the identified terminal nodes' performances meet an accuracy level (ε, the error rate). In this case, it is not necessary to specify the depth of the tree, but it can cause overfitting of the model.

The algorithm results are represented in Fig. 2.4, where $L = 3$, $K = 5$, and $\pi = [0, 0, 1, 1, 1]$. In Fig. 2.5 a three-dimensional example is shown. The function

$$y = \frac{\sin\left(\sqrt{x_1^2 + x_2^2 + \varepsilon}\right)}{\sqrt{x_1^2 + x_2^2 + \varepsilon}} \tag{2.22}$$

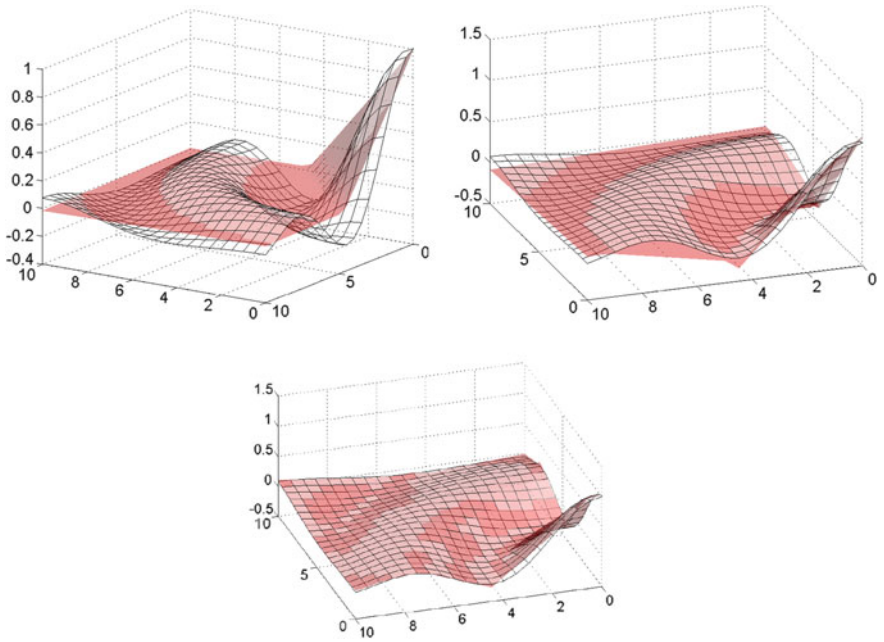

Fig. 2.5 Modeling a 3D function with hinging hyperplane-based tree

has been approximated by a hinging hyperplane-based tree. In Fig. 2.5 it is shown how the approximation becomes much more smooth with applying one, two, and four levels and greedy building method.

The generation of the binary tree structured model is not the only thing important; to construct a greedy algorithm and to measure the identified model, node performance must be determined during the identification procedure. It can be defined in different ways:

- *Modeling performance of the nodes*

  The well-known regression performance estimators can be used for node performance measurement; in this work, root mean squared prediction error (RMSE) was used:

$$RMSE = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(y_k - \hat{y}_k)^2}. \tag{2.23}$$

- *Condition of the nodes*

  It is described that with two prototype clusters ($c = 2$) and apriori knowledge (*constraints*), the FCRM method is able to identify hinging hyperplanes; hence, membership degree $\mu_{i,k}$ has information at a node about how many data points are going to the slitted prototypes. Comparing this data with the information about the hinge-based node splitting rule (how many data samples are described by the θ_1, θ_2 parameter vectors), we can assign a certain condition (ρ) to a node:

$$\rho_n = 1 - \frac{\|m_1 - \sum_{k=1}^{N_n} \mu_{1,k}\|}{\sum_{i=1}^{2}\sum_{k=1}^{N_n} \mu_{i,k}}, \tag{2.24}$$

  where m_1 is the cardinality based on θ^+, while N_n represents the number of samples at node π_i.

We can consider ρ as a measurement of the FCRM hinging hyperplane identification perfection. The closer ρ is to 1, the better the identification. The hinge does not "override" the $\mu_{i,k}$ membership degrees. This measure is very similar to the one that was introduced in [49] and was used for identifying parameter similarity. In Figs. 2.6 and 2.7 RMSE and ρ results of identifying Eq. 2.22 node by node (axis x) with the depth of four levels can be seen. In Fig. 2.6, the non-greedy case is examined, while Fig. 2.7 shows the performance of the greedy algorithm. It is apparent that for tree-building purposes, cardinality-based splitting is a very good approach.

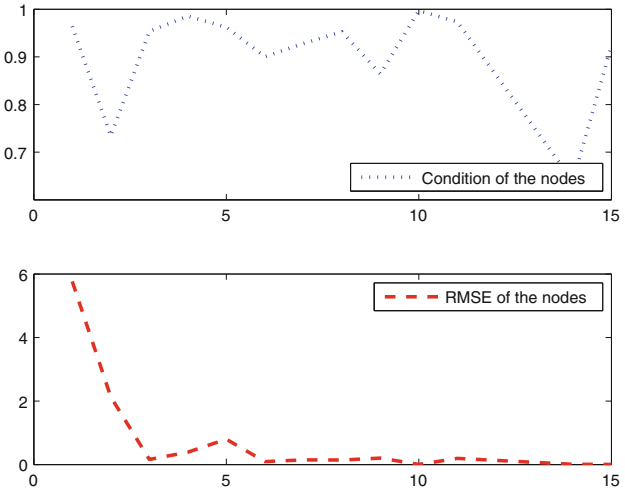

Fig. 2.6 Node by node ρ and RMSE results for non-greedy tree building

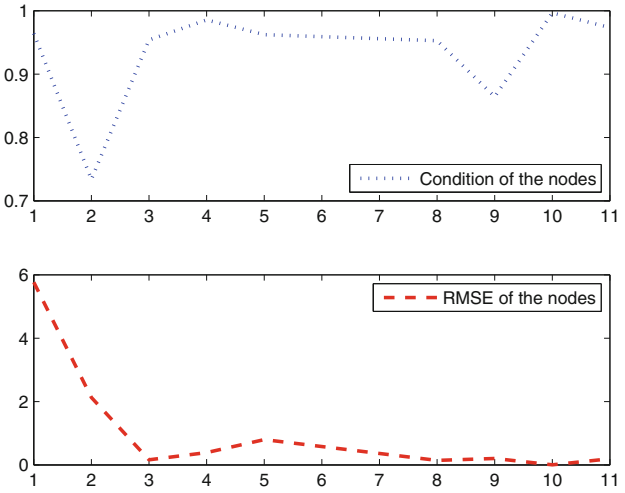

Fig. 2.7 Node by node ρ and RMSE results for greedy tree building

2.3 Application Examples

Accuracy and transparency of the proposed algorithm are shown based on multiple datasets, two real life and two synthetic ones, followed by examples from the area of dynamic system identification.

2.3.1 Benchmark Data

All datasets have been used before, most of them have originated from well-known data repositories. Performance of the models is measured by the root mean squared prediction error (RMSE; see Eq. 2.23).

Real life datasets:

- *Abalone* Dataset from UCI machine learning repository[1] used to predict the age of abalone from physical measurements. Contains 4,177 cases with eight attributes (one nominal and seven continuous).
- *Kin8nm* Data containing information on the forward kinematics of an eight link robot arm from the DVELVE repository. Contains 8,192 cases with eight continuous attributes.

Synthetic datasets:

- *Fried* Artificial dataset used by Friedman [50] containing ten continuous attributes with independent values uniformly distributed in the interval [0, 1]. The value of the output variable is obtained with the equation:

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \sigma(0, 1). \tag{2.25}$$

- *3DSin* Artificial dataset containing two continuous predictor attributes uniformly distributed in interval $[-3, 3]$, with the output defined as

$$y = 3\sin(x_1)\sin(x_2). \tag{2.26}$$

3,000 data points were generated using these equations.

For the robust testing of the performance of the model-building algorithm, ten-fold cross-validation method is utilized with data normalized to zero mean and unit variance. Table 2.1 shows the performance on these datasets compared to the results of other algorithms and can be found in [38]. Moreover, Table 2.2 contains the min, mean, and max error rates of the ten-fold cross-validation results for the hinging hyperplane algorithm with the calculated standard deviation values. The comparative algorithms are fuzzy-based (*FRT-Fuzzy Regression Tree, FMID-Fuzzy Model*

[1] FTP address: ftp://ftp.ics.uci.edu/pub/machine-learning-databases/.

Table 2.1 Comparison of RMSE results of different algorithms

Data	Sample	HH	CART	FMID	FRT
Fried	Train	0.87	0.84	2.41(4)	0.69
	Test	0.92(8)	2.12(495.6)	2.41(12)	0.7(15)
3Dsin	Train	0.17	0.09	0.50(4)	0.18
	Test	0.18(11)	0.17(323.1)	0.31(12)	0.18(12)
Abalone	Train	2.62	1.19	2.20(4)	2.18
	Test	2.88(8)	2.87(664.8)	2.19(12)	2.19(4)
Kinman	Train	0.15	0.09	0.20(4)	0.15
	Test	0.16(6)	0.23(453.9)	0.20(12)	0.15(20)

Numbers in brackets are the number of models

Table 2.2 10-fold cross validation report for hinging hyperplanes based tree

Data	Sample	MIN	MEAN	MAX	Standard dev.
Fried	Train	0.5822	0.8677	1.2107	0.227
	Test	0.6226	0.9208	1.2673	0.2337
3Dsin	Train	0.0906	0.1741	0.3162	0.0714
	Test	0.0838	0.178	0.342	0.0801
Abalone	Train	2.3496	2.6241	2.9256	0.1532
	Test	2.3242	2.8803	3.451	0.3445
Kinman	Train	0.1433	0.1515	0.1595	0.0054
	Test	0.1464	0.1579	0.1729	0.0092

Identification) and classical regression tree-based (*CART-Classification and Regression Tree*). It can be concluded that the performance of the introduced algorithm is in line with even the other methods, that of with a moderate number of terminal nodes in the identified model tree. Results are consistent; even for the worst performance of the ten-fold cross-validation.

2.3.2 Application to Dynamical Systems

The following subsection shows results on the identification first-order nonlinear dynamic system and describes performance of the proposed technique in model predictive control.

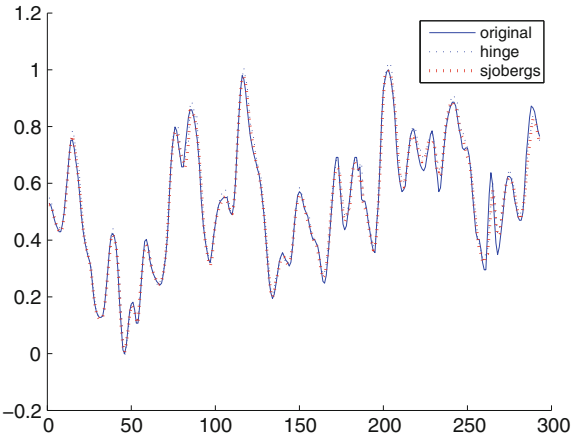

Fig. 2.8 Identification of the Box-Jenkins gas furnace model with hinging hyperplanes

2.3.2.1 Identification of the Box-Jenkins Gas Furnace

The well-known Box-Jenkins furnace data benchmark is used to illustrate the proposed modeling approach and to compare its effectiveness with other methods. The data set consists of 296 pairs of input-output observations taken from a laboratory furnace with a sampling time of nine seconds. The process input is the methane flow rate and the output is the percentage of CO_2 in the off-gas. A number of researchers concluded that a proper structure of a dynamic model for this system is

$$y(k+1) = f(y(k), u(k-3)). \tag{2.27}$$

The approximation power of the model can be seen in Fig. 2.8 and Table 2.3. Comparing its results with those of other techniques in [51] can conclude that the modeling performance is in line with that of other techniques with a moderate number of identified hinging hyperplanes.

So far, *a general nonlinear modeling technique* was presented and a new identification approach was given for hinging hyperplane-based nonlinear models: $\widehat{y} = f(\mathbf{x}(k), \theta)$, where $f(.)$ represents the hinging hyperplane-based tree structured model and $\mathbf{x}(k)$ represents the input vector of the model. To identify a discrete-time *input-output model* for a dynamical system, the dynamic model structure has to be

Table 2.3 RMSE results of the generated models

Method	Training	Testing	Free run	HH ♯
Proposed technique	0.0266	0.0311	0.0374	4
Sjoberg model	0.0336	0.0342	0.0351	4

chosen or determined beforehand. A possible structure often applied is the nonlinear autoregressive model with exogenous input (NARX), where the input vector of the model $\mathbf{x}(k)$ contains the delayed inputs and outputs of the system to be modeled [32]. In several practical cases a simpler and more specific model structure can be used to approximate the behavior of the system, which fits better the structure of the system. Therefore, it can be an advantage for the identification approach (models with simpler structures can be identified), and this model can be more accurate. One such special case of the NARX model is the Hammerstein model, where the same static nonlinearity f is defined for all of the delayed control inputs (for the sake of simplicity, SISO models are considered):

$$\widehat{y} = \sum_{i=1}^{n_a} a_i y(k-i) + \sum_{j=1}^{n_b} b_j, f(u(k-j)) \tag{2.28}$$

where $y()$ and $u()$ are the output and input of the system, respectively, and n_a and n_b are the output and input orders of the model. The parameters of the blocks of the Hammerstein model (static nonlinearity and linear dynamics) can be identified by the proposed method simultaneously if the same linear dynamic behavior can be guaranteed by all of the local hinging hyperplane-based models. It can be done in an elegant way utilizing the flexibility of the proposed identification approach: global constraints can be formulated for the a_i and b_j parameters of the local models (for a detailed discussion on what constraints have to be formulated, see [32]). In the following, the hinging hyperplane modeling technique is applied on a Hammerstein type system. It will be shown why it is an effective tool for the above-mentioned purpose.

2.3.2.2 Application to Model Predictive Control

The modeling of a simulated water heater (Fig. 2.9) is used to illustrate the advantages of the proposed hinging hyperplane-based models. The water flows through a pair of metal pipes containing a cartridge heater.

The outlet temperature, T_{out}, of the water can be varied by adjusting the heating signal, u, of the cartridge heater (see [32] or Appendix D for details). The performance of the cartridge heater is given by:

$$Q(u) = Q_M \left[u - \frac{\sin(2\pi u)}{2\pi} \right] \tag{2.29}$$

where Q_M is the maximal power and u is the heating signal (voltage). As the equation above shows, the heating performance is a static nonlinear function of the heating signal. Hence, the Hammerstein model is a good match to this process. The aim is to construct a dynamic prediction model from data for the output temperature (the dependent variable, $y = T_{out}$) as a function of the control input: the heating

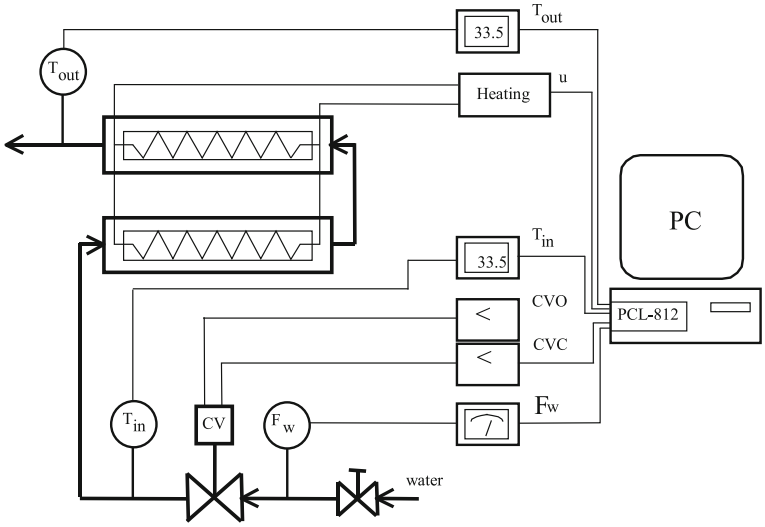

Fig. 2.9 The water heater

signal. The parameters of the Hammerstein model were chosen as $n_a = n_b = 2$. The performance of this modeling technique will be compared to that of linear and feedforward neural network models. The modeling performances can be seen in Table 2.4. Modeling errors were also calculated based on RMSE (see Eq. (2.23)). In this example, a hinging hyperplane function-based tree with four leaves was generated. For robust testing of the model-building algorithm performance, ten-fold cross-validation method was used. For comparison, a feedforward neural net and linear model was also trained and tested using the same data. The neural net contains one hidden layer with 4 neurons using tanh basis functions. As can be seen from the results, the training and test errors are comparable with the errors of the proposed method. A very rigorous test of NARX models is free run simulation because the errors can be cumulated. It can be also seen in Fig. 2.10 that the identified models (the proposed ones, linear models and the neural nets) perform very good also in free run simulation (the system output and the simulated ones can hardly be distinguished). Although the neural net seems to be more robust in this example, the proposed hinging hyperplane model is much more interpretable than the neural net [1]. This confirms that both

Table 2.4 RMSE results of the generated models

Method	Training	Testing	Free run
Linear model	0.0393	0.0449	0.387
Neural network	0.0338	0.0403	0.356
Proposed method	0.0367	0.0417	0.359

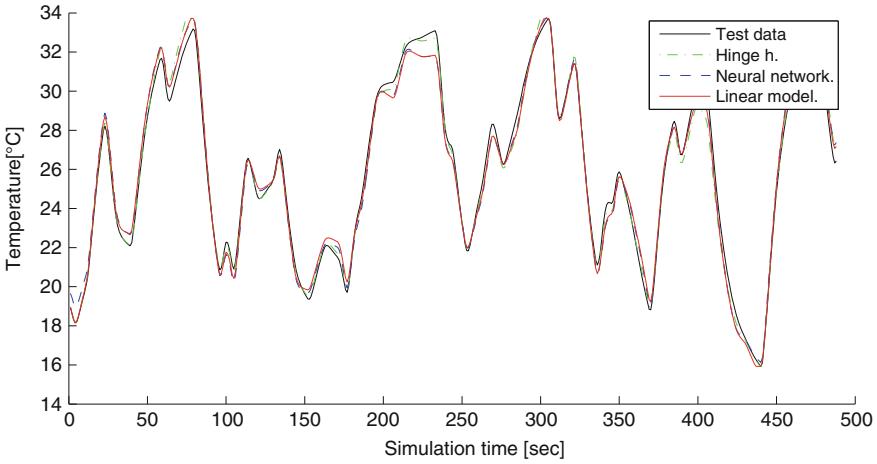

Fig. 2.10 Free run simulation of the water heater (proposed hinging hyperplane model, neural network, linear model)

the proposed clustering-based constrained optimization strategy and the hierarchical model structure have advantages over the classical gradient-based optimization of global hinging hyperplane models.

In the following, this model will be applied for model predictive control. Details of model-based control of fuzzy and operating regime models can be found in [41, 42]. Among the wide range of possible solutions, a model predictive controller (MPC) was designed. Figure 2.11 shows the structure of an MPC controller.

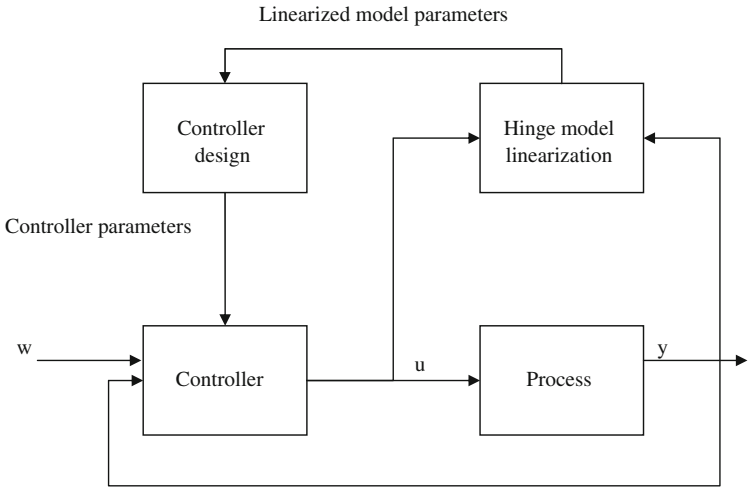

Fig. 2.11 Structure of the MPC controller

Real-time control needs low computational complexity. Hence a time-varying linear MPC has been designed based on time-varying parameters of a linear model extracted at every time instant from the regression tree. This scheme has been widely studied and similarities to the nonlinear optimization based control solutions including convergence have also been shown. In [32] it was shown that there are two ways to obtain a linear model from a nonlinear operating regime-based (fuzzy) model. Taylor expansion-based linearization assumes global interpretation of the model, while the linear parameter varying (LPV) model extraction approach considers the model as an interpolating system between local linear time-invariant (LTI) models were the dynamic effect of the interpolation is negligible. Fortunately, thanks to hinging hyperplane models and tree-structured representation, the proposed model perfectly supports the interpretation, that hinging hyperplanes define local linear models and their operating regimes. Since these local models do not overlap, the negative effect of the interpolating functions do not have to be taken into account.

The classical model predictive controller computes an optimal control sequence by minimizing the following cost quadratic cost function:

$$J\left(H_p, H_c, \lambda\right) = \sum_{j=1}^{H_p} \left(w\left(k+j\right) - \hat{y}\left(k+j\right)\right)^2 + \lambda \sum_{j=1}^{H_c} \Delta u^2\left(k+j-1\right), \quad (2.30)$$

where $\hat{y}\left(k+j\right)$ denotes the predicted process output, H_p is the maximum costing or prediction horizon, H_c is the control horizon, and λ is a weighting coefficient. According to the receding horizon principle, only the first element of the optimized control sequence is applied $u(k)$, and this optimization is performed at every time instant. This scheme allows real-time control, provides feedback on model errors, handles unmeasured disturbances, and supports the previously mentioned iterative liberalization scheme. Details about the convergence and possible extensions of this control scheme can be found in [33].

The key equation of MPC is the prediction of the model:

$$\hat{\mathbf{y}} = \mathbf{S}\Delta\bar{\mathbf{u}} + \mathbf{p}, \quad (2.31)$$

where the model prediction equation is given in its vector-based form as $\Delta\bar{\mathbf{u}} = [\Delta u(k), \ldots, \Delta u(k+H_c)]$, and $\mathbf{p} = \left[p_1, p_2, \ldots, p_{Hp}\right]$ and $\hat{\mathbf{y}} = [\hat{y}(k+1), \ldots, \hat{y}(k+H_p)]$ and the $\mathbf{S}$ containing the parameters of a step-response model is an $\left(H_p\right) \times H_c$ matrix with 0 entries $s_{i,j}$ for $j - i > 1$:

$$\mathbf{S} = \begin{bmatrix} s_1 & 0 & 0 & 0 \\ s_2 & s_1 & 0 & 0 \\ \vdots & & \ddots & \\ s_{H_p} & s_{H_p-1} & \cdots & s_{H_p-H_c} \end{bmatrix}. \quad (2.32)$$

When constraints are considered, the minimum of the cost function can be found by quadratic optimization with linear constraints:

$$\min_{\bar{\mathbf{u}}} \left\{ (\mathbf{S}\Delta\bar{\mathbf{u}} + \mathbf{p} - w)^T (\mathbf{S}\Delta\bar{\mathbf{u}} + \mathbf{p} - w) + \lambda \Delta\bar{\mathbf{u}}^T \Delta\bar{\mathbf{u}} \right\}, \tag{2.33}$$

$$\min_{\bar{\mathbf{u}}} \left\{ \frac{1}{2} \Delta\bar{\mathbf{u}}^T \mathbf{H} \Delta\bar{\mathbf{u}} + \mathbf{d}\Delta\bar{\mathbf{u}} \right\},$$

with $\mathbf{H} = 2\left(\mathbf{S}^T\mathbf{S} + \lambda\mathbf{I}\right)$, $\mathbf{d} = -2\left(\mathbf{S}^T(\mathbf{w} - \mathbf{p})\right)$, where $\mathbf{I}$ is an $(H_c \times H_c)$ unity matrix.

The constraints defined on u and Δu can be formulated with the following inequality:

$$\begin{pmatrix} \mathbf{I}_{\Delta\bar{\mathbf{u}}} \\ -\mathbf{I}_{\Delta\bar{\mathbf{u}}} \\ \mathbf{I}_{H_c} \\ -\mathbf{I}_{H_c} \end{pmatrix} \Delta\bar{\mathbf{u}} \leq \begin{pmatrix} \mathbf{u}_{\max} - \mathbf{I}_{\bar{\mathbf{u}}} u(k-1) \\ -u_{\min} + \mathbf{I}_{\bar{\mathbf{u}}} u(k-1) \\ \Delta\mathbf{u}_{\max} \\ -\Delta u_{\min} \end{pmatrix}, \tag{2.34}$$

where $\mathbf{I}_{H_c}$ and $\mathbf{I}_{\bar{\mathbf{u}}}$ is an $H_c \times H_c$ unity matrix, $\mathbf{I}_{\Delta\bar{\mathbf{u}}}$ is an $H_c \times H_c$ lower triangular matrix with all elements equal to 1, and $\Delta\mathbf{u}_{\min}, u_{\max}, \mathbf{u}_{\min}, \mathbf{u}_{\max}$ are H_c-vectors, with the constraints $\Delta u_{\min}$, $\Delta u_{\max}$, $u_{\min}$, $u_{\max}$, respectively.

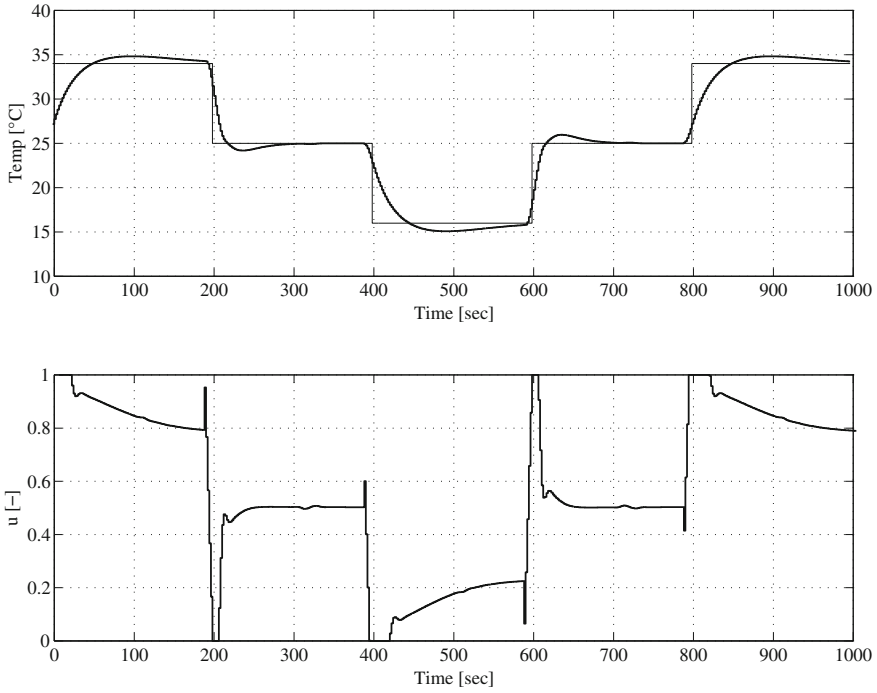

Fig. 2.12 Performance of the MPC based on linear model

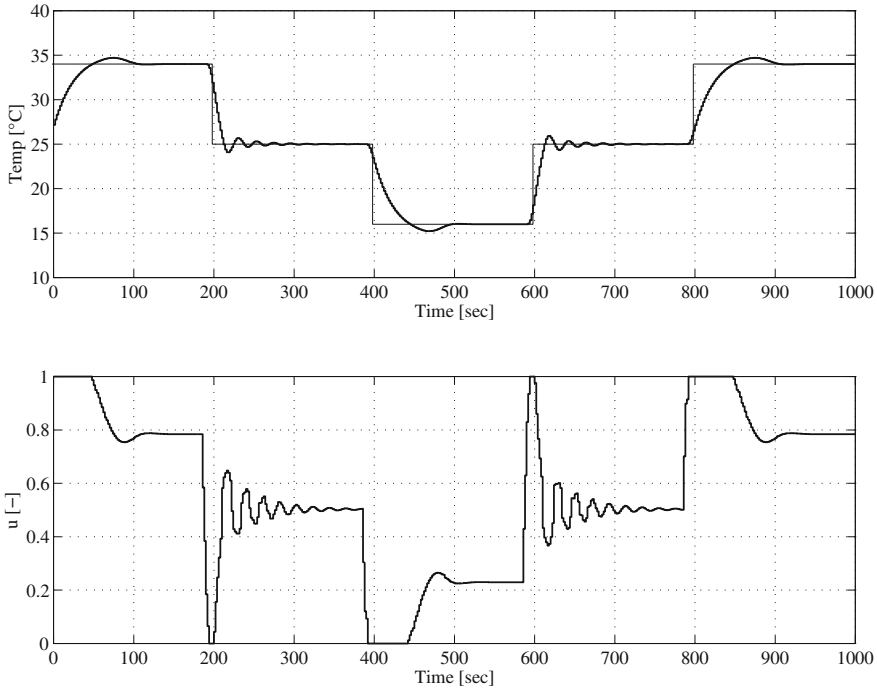

Fig. 2.13 Performance of the MPC based on neural network model

To handle modeling error, the MPC is applied in the well-known internal model control (IMC) scheme where the setpoint of the controller is shifted by the filtered modeling error. For this purpose, a first-order linear filter is used:

$$e_{mf}(k) = \alpha e_m(k) + (1-\alpha)\, e_{mf}(k-1)\,, \tag{2.35}$$

where $0 \leq \alpha < 1$ is determined such that a compromise between performance and robustness is achieved. Effective suppressing of the steady-state modeling error can be achieved by proper tuning of this filter. The best parameters are found for the controller: $H_p = 9$, $H_c = 2$, $\lambda = 20$ and $\alpha = 0.95$. Simulation results for the hinging hyperplane based model, the affine neural network model and the linear model are shown in Figs. 2.12, 2.13 and 2.14.

At the operation region edges, the MPC based on the linear model resulted in undesirable overshoots and undershoots. This is a direct consequence of a bad estimation of the nonlinear gain of the system in these regions. This overestimation of the system gain by the linear model is also seen in the sluggish control action. In contrast, the MPC based on the nonlinear models shows superior performance over the whole operating region. Among these, the MPC based on the hinging hyperplane model results in the smallest overshoot with the fastest change in the control signal.

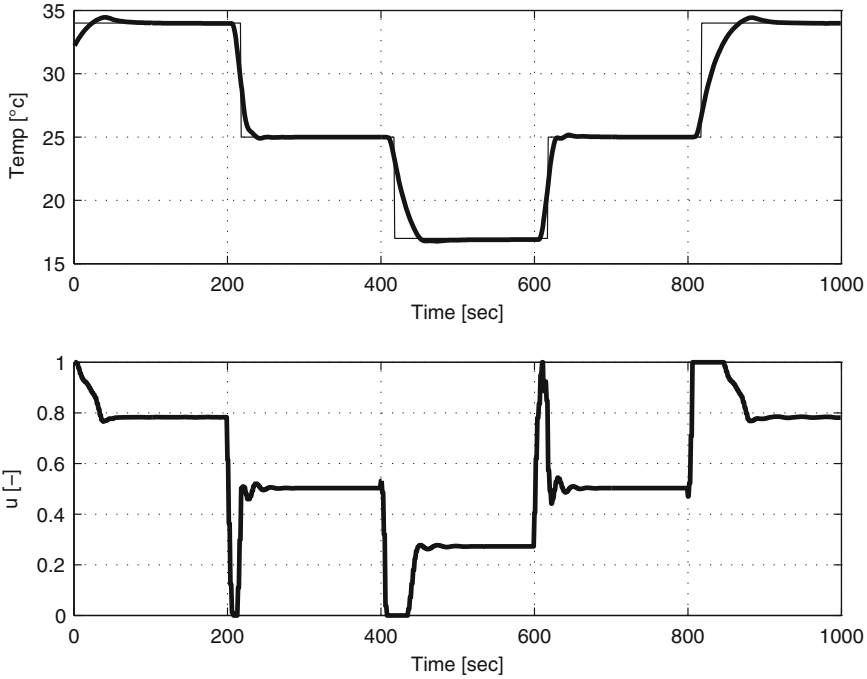

Fig. 2.14 Performance of the MPC based on hinging hyperplane

Table 2.5 Simulation results (SSE, sum squared tracking error; CE, sum square of the control actions)

The applied model in GPC	SSE	CE
Linear model	1085	1.61
Neural network model	956	1.39
Hinging hyperplane model	966	0.58

Notice also that the oscillatory behavior of the neural network model-based MPC is due to the bad prediction of the steady-state gain of the system around the middle region. However, as can be seen from Table 2.5, both nonlinear models achieved approximately the same summed squared tracking error (SSE), although a smaller control effort (CE) was needed for the hinging hyperplane-based MPC.

2.4 Conclusions

A novel framework for hinging hyperplane-based data-driven modeling has been developed. Fuzzy c-regression clustering can be used to identify the parameters of two hyperplanes. A hierarchical regression tree is obtained by the recursive cluster-

ing of the data. The complexity of the model is controlled by the proposed model performance measure. The resulting piecewise linear model can be effectively used to represent nonlinear dynamical systems. The resulting linear parameter varying (LPV) model can be easily utilized in model-based control.

To illustrate the advantages of the proposed approach, benchmark datasets were modeled and a simulation example presented for the identification and model predictive control of a laboratory water heater.

The results show that, with the use of the proposed modeling framework, accurate and transparent nonlinear models can be identified since the complexity and the accuracy of the model can be easily controlled. The local linear models can be easily interpreted and utilized to represent operating regimes of nonlinear dynamical systems. Based on this interpretation, effective model-based control applications can be designed.

Chapter 3
Interpretability of Neural Networks

A neural network is a black-box model, so it doesn't reveal any information about the identified system. It is a challenging task to open up this box to support model-building procedures. However, based on the extracted information, model reduction and visualization could be done on the base model. The key idea is that the neural networks can be transformed into a fuzzy rule base where the rules can be analyzed, visualized, interpreted and even reduced.

Section 3.1 shows how NNs work. This description is mainly based on [31]; for a detailed discussion, see [1]. Having the main concepts set for NNs, Sect. 3.2 gives a brief introduction and overview about the applied NN transformation method, which means the basis for model reduction and visualization. Section 3.3 contains a combined approach used in this book to get a reduced rule-based model from NNs. Section 3.4 gives an overview of some NN visualization methods, and proposes a new technique to measure the similarity of neurons, which gives the basis of the visualization approach. In Sect. 3.5 some illustrative examples are given, and Sect. 3.6 concludes the chapter.

3.1 Structure of Neural Networks

The systematic study of continuous functions started in the nineteenth century. The complexity of this function class was demonstrated by Weierstrass' famous example; *the everywhere continuous but nowhere differentiable real function* [52].

Theoretical foundations for function approximation were also aided by Weierstrass: he showed that a*ny continuous function on a closed real interval could be uniformly approximated by polynomials* [53]. Another question was the theory of multivariate continuous function approximation. Around 1900, Hilbert suspected the existence of continuous multivariate functions, which cannot be approximated by arbitrarily chosen continuous univariate functions [54]. Counterexamples for Hilbert's conjecture arose only in 1957. Ternary continuous functions prepared by Arnold [55]. followed by constructive example from Kolmogorov were prepared for the approximation of multivariate continuous functions by continuous univariate functions [56].

T. Kenesei and J. Abonyi, *Interpretability of Computational Intelligence-Based Regression Models*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-21942-4_3

Later, the Kolmogorov method was simplified in the early 1960s (e.g., Sprecher [57] and Lorentz [58]); it has become the theoretical basis of universal approximators of continuous functions. It was revealed in the 1980 and 1990s that feedforward multilayer neural networks [59–62] and fuzzy systems constructed by Zadeh [63] are universal approximators [64–66], although the limitations of the approximation capabilities of systems constructed from finite components became clear [67–69].

Around the mid-1950s Marshall Harvey Stone further generalized Weierstrass' statements [70, 71], so the Stone-Weierstrass theorem is mentioned in respect to neural networks [72, 73].

3.1.1 McCulloch-Pitts Neuron

McCulloch and Pitts in 1943 developed a simple mathematical model for a neuron (Fig. 3.1).

The McCulloch-Pitts neuron has multiple inputs and a single output. Each of the inputs has an associated weight. Weighted sum of the inputs is passed through a nonlinearity to the output of the neuron as follows:

$$y = f\left(\sum_{i=1}^{N} \mathbf{w}_i \mathbf{x}_i\right), \quad \begin{cases} 0 = z < 0 \\ 1 = z \geq 0 \end{cases} \tag{3.1}$$

where x_i are the inputs and y is the output of the neuron, $f(z)$ is the nonlinear activation function in the form of the step function given above, and w_i are the strengths of the connections or weights. A multilayer neural network is a network of neurons bunched together in a multiple layer network. A feedforward neural network has one input layer, one output layer and a number of hidden layers between them. Normally we use neural networks with one hidden layer.

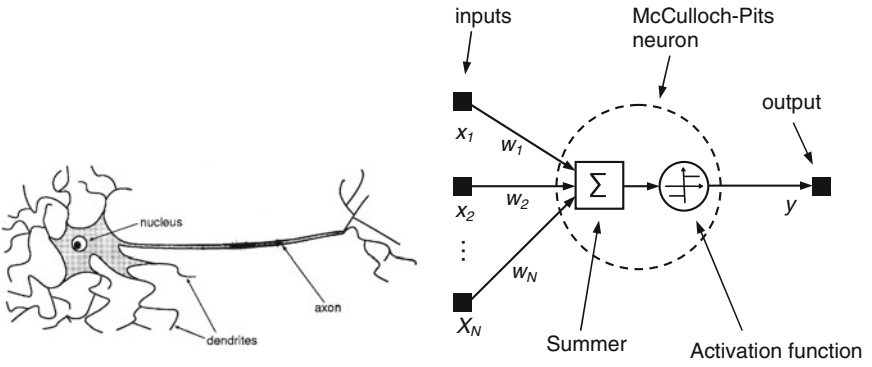

Fig. 3.1 A biological neuron and its model (McCulloch-Pitts neuron)

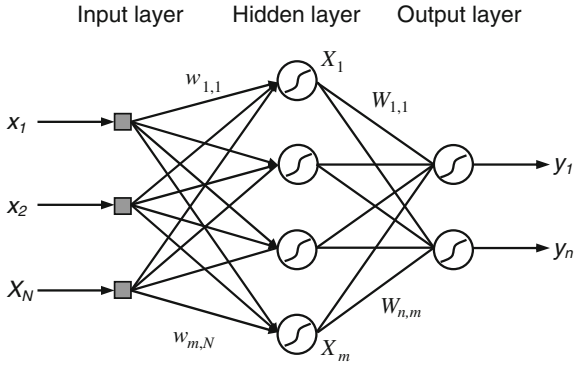

Fig. 3.2 A multi-layer neural network with one hidden layer of neurons

Input-layer neurons distribute inputs x_k to the weights w_{jk} of the hidden layer. In the neurons of the hidden layer, first the weighted sum of the inputs is computed:

$$h_j = \sum_k w_{jk} \mathbf{x}_k. \tag{3.2}$$

A nonlinear transfer function (also known as activation function and squashing function) is applied to the result to calculate each processing element's output:

$$\mathbf{X}_j = f\left(\sum_k w_{jk} \mathbf{x}_k\right). \tag{3.3}$$

The transfer function adds nonlinearity and stability to the network. The final output (see Fig. 3.2) is calculated as

$$Y_i = f\left(\sum_j \mathbf{W}_{ij} f\left(w_{jk} \mathbf{x}_k\right)\right). \tag{3.4}$$

This model is very general. It has been shown that with one hidden layer a network can describe any continuous function (if there are enough hidden units), and that with two hidden layers it can describe any function. A detailed description of neural network structures, utilization and activation function types can be found in [1, 17].

Based on these model types, our objective is to prepare a tool where data, prior knowledge, prediction and extracted information (see Fig. 3.3) form an integrated framework to help model-building procedures with interpretability, visualization and reduction of multilayer perceptron (MLP, where usually one hidden layer is applied)-type neural networks with a logistic hidden activation function.

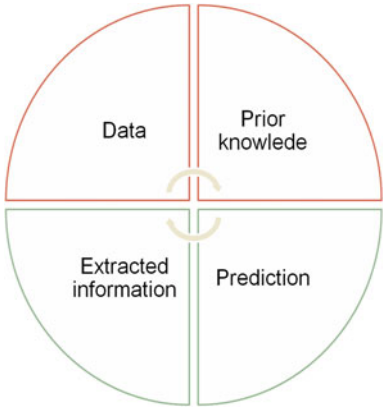

Fig. 3.3 Modeling framework

3.2 NN Transformation into Rule-Based Model

A possible strategy for 'opening' an NN is to convert it into a rule-based model. These 'linguistically sound' rules are often fuzzy if-then rules, and are close to human thinking: IF *a set of conditions is satisfied*, THEN *a set of consequences is inferred*. Fuzzy logic provides a tool to process uncertainty; hence fuzzy rules represent knowledge using linguistic labels instead of numeric values; thus, they are more understandable for humans and may be easily interpreted [31]. If NNs can be transformed into rules, then it becomes possible to overlook and validate the trained NN, and build a priori knowledge into the network. The crucial question is about what the connection is between the several types of neural networks and fuzzy rule-based systems.

Under some conditions, the equivalence of normalized radial basis function networks (RBFs) and Takagi-Sugeno fuzzy models can be obtained [1]. However, in this book, multilayer perceptron (MLP)-type neural networks with logistic hidden activation functions are used (in the following, the notation NN will be used for MLP-type networks). An approach for NNs with a tanh activation function is presented in [74] for function approximation purposes, but it should be noted that it is an approximation: the rule-based model is not identical to the original trained NN; therefore information transfer in the 'opposite' direction, i.e., from the rule base to the NN, can be problematic. An interesting result was given in [31], where the equality of NNs with logistic activation functions and a certain type of fuzzy rule-based model called fuzzy additive system (FAS) was proven. For that purpose, a new fuzzy logic operator had to be introduced. Because of the equality (which is stronger than equivalence), if a method can be applied on an FAS for a certain purpose (e.g., rule base reduction), then it is also applicable to the NN, and vice versa.

3.2.1 Rule-Based Interpretation of Neural Networks

In the following, this equality relation is discussed based on [31]. FAS employs rules in the following form:

$$\mathbf{R}_{jk}: \textbf{If } x_1 \; is \; A_{jk}^1 \; and \ldots and \; x_n \; is \; A_{jk}^n \textbf{ then } y_k \; is \; \delta_{jk}(x_1, \ldots, x_n), \quad (3.5)$$

where $\delta_{jk}(x_1, \ldots, x_n)$ is a linear function of the inputs. In FASs, the inference engine works as follows: for each rule, the fuzzified inputs are matched against the corresponding antecedents in the premises, giving the rule's firing strength. It is obtained as the t-norm (usually the minimum operator) of the membership degrees on the rule's if-part. The overall value for output y_k is calculated as the weighted sum of relevant rule outputs. Let us suppose we have multi-intput single-output fuzzy rules, l_k of them for the kth output. Then y_k is computed as

$$y_k = \sum_{j=1}^{l_k} \beta_{jk} \delta_{jk}(x_1, \ldots, x_n), \quad (3.6)$$

where β_{jk} is the firing strength of the jth rule for kth output.

To decompose the multivariate logistic function to form the rule antecedents in the form of Eq. 3.5 with *univariate* membership functions, a special logic operator has to be used instead of *and*: interactive-or or i-or:

$$a * b = \frac{(ab)}{(1-a)(1-b)+ab}. \quad (3.7)$$

To get a clearer idea of i-or behavior, see Fig. 3.4, which represents the surface defined by the i-or operator. Using this $*$ operator, the interpretation of NNs whose hidden neurons have biases is as follows. It can be checked that

$$f_A\left(\sum_{i=1}^{n} x_i w_{ij} + \tau_j\right) = f_A\left(x_1 w_{1j} + \tau_j'\right) * \cdots * f_A\left(x_n w_{nj} + \tau_j'\right), \quad (3.8)$$

where $\tau_j' = \tau_j / n$ and the first term corresponds to the fuzzy proposition "$\sum_{i=1}^{n} x_i w_{ij} + \tau_j$ is A". Likewise, $f_A\left(x_i w_{ij} + \tau_j'\right)$ corresponds to proposition "$x_i w_{ij} + \tau_j'$ is A" or, in a different form, "$x_i w_{ij}$ is $A - \tau_j'$". Hence, the bias term means a sheer translation. The A_{jk}^i fuzzy sets have to be redefined to account for both the weight w_{ij} and the bias τ_j'.

Their membership function is defined by (see Fig. 3.5 for a better explanation):

$$\mu_{A_{jk}^i}(x) = \mu_A\left[\left(x + \tau_j'\right) * w_{ij}\right]. \quad (3.9)$$

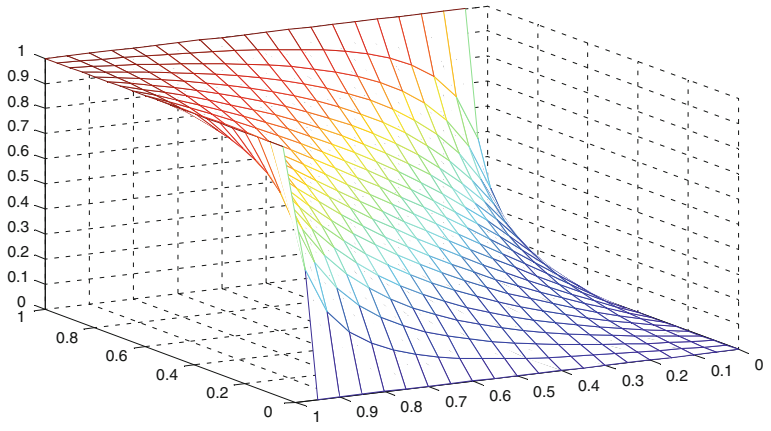

Fig. 3.4 Interactive or operator

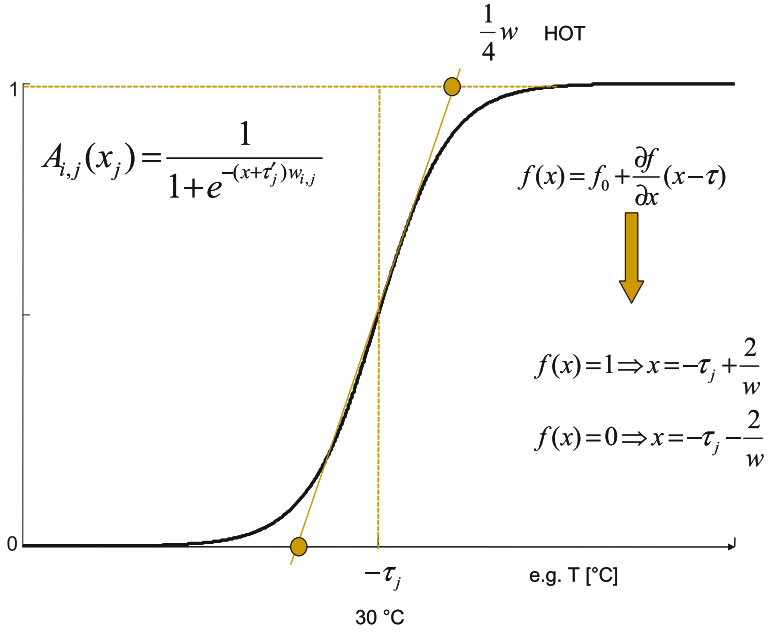

Fig. 3.5 Interpretation of the activation function

Based on that, the fuzzy rules extracted from the trained NN are

$$\mathbf{R}_{jk}: \textbf{If } x_1 \; is \; A^1_{jk} \; * \ldots and * \; x_n \; is \; A^n_{jk} \textbf{ then } y_k = \delta_{jk}. \tag{3.10}$$

An interesting and useful application possibility is to initialize the NN on the basis of a priori knowledge. Initialization is a crucial question for NNs because there are often a huge number of parameters and the cost function has numerous local minima. The most often applied local (gradient-based) search techniques may get

trapped in a local minimum. To avoid that problem, a possible approach to use is the multi-start method, i.e. to train the NN from several different (random) initial points. Other solutions can be based on evolutionary algorithms; see [75]. The flexibility of evolutionary algorithms makes possible direct rule extraction from trained NNs (however, only crisp rules, and for classification problems), as [76] shows. However, all of these latter methods have high computational demand. The initialization using prior knowledge-based if-then rules has another advantage as well: it combines the user's experience with the learning capability of the NN.

3.3 Model Complexity Reduction

In this section we focus on the combination of existing model reduction techniques with the previously presented rule-based model extraction method. An interesting solution to NN reduction is the following: the complexity of the model is penalized, and it is built into the training procedure. The method proposed in [77] uses a cost function that consists of two terms: one for the NN accuracy (such as the mean square error) and one related to the NN complexity (number and magnitude of parameters). However, determination of their weights or relative importance is problematic. A weighting factor is introduced and several NNs should be trained with different weighting parameters. To compare the trained NNs and choose the best one, [77] applied the predicted square error measure.

In the case of MLP networks and FAS systems, classical OLS (see Appendix B for further details) can be applied on FAS systems to rank the rules since the parameters of the trained NN are fixed. However, OLS is formulated as a MISO technique. If the NN has more than one output, then the outputs can be evaluated individually one by one. In this case (using the notation of OLS Eqs. (B.1)–(B.3)), y is the kth network output, the regressors z_i are the outputs of the hidden neurons, and the parameters θ_j corresponds to the weights from the jth hidden neuron to the kth output neuron β_{jk}. This approach was directly applied on NNs in [30], and it was shown that an analogous method can be applied to the subset selection of the original network inputs. In this case, in Eqs. B.1–B.3, y is the output of the kth hidden neuron, the regressors z_i are the inputs of the network, and the parameters θ_j correspond to the weights from the jth input neuron to the kth hidden neuron w_{jk}. Other NN pruning can also be considered, e.g., optimal brain damage [26] or optimal brain surgeon [78], and it should be emphasized that these methods can directly be applied on FAS systems as well. The application examples in Sect. 3.5 show that it can be very effective when a model reduction technique and rule base extraction from NN is applied together.

Note that ordering the neurons by OLS estimated error reduction ratios reveals the unnecessary neurons (the importance of the extracted rules) in the hidden layer, because neurons with low error reduction ratios are insignificant for the appropriate model. The equality of FAS and NN was proven in [31] and was discussed also in Sect. 3.2; the OLS ranking means a reduction based on the *consequent of the fuzzy rule*.

It should be noted that the applied i-or operator in the extracted fuzzy rules does not belong to the commonly applied fuzzy t-norms or t-conorms. However, it would be interesting to test the extracted fuzzy rules with common fuzzy logic operators, and maybe recompute the output weights (which can easily be done because the model is linear with respect to these parameters). Our presumption is that the crisper the activation functions are (f_A), the less the difference is between the modeling performances of the original and the modified FASs that use classical fuzzy logic operators. For that purpose, numerous tests have to be completed in the future. If this guess proves true, the cost function for NN training can be modified to get 'crisper' activation functions.

3.4 Visualization of Neural Networks

In this section, a new technique for the visualization of neural networks is proposed. First, methods are discussed that can directly be applied on NNs. Second, a new approach is presented to detect the redundant neurons based on their similarity. This method exploits the equality of NNs and FASs because it is based on the similarity of fuzzy membership functions.

The output of hidden neurons z_j can be seen as an h-dimensional vector that represents the range the neurons work in. If a 'hidden variable' z_j is close to 0 or 1, the neuron is saturated. If a hidden neuron gives values near 0 or 1 for almost all inputs, and hence does not fire or fires all the time, it is useless for the problem. The distribution of these h-dimensional data can represent the NN behavior for a human expert. Unfortunately, in several cases there is a need for more than two or three hidden neurons. In these cases, a projection or dimensionality reduction technique has to be used. Principal Component Analysis (PCA) is a linear technique; therefore, the information loss may be more than the admissible level. Other (topology or distance preserving) projection techniques such as Multidimensional Scaling, Sammon method, Isomap or Locally Linear Embedding can be used for that purpose. For more details, see [32] and the references therein.

However, there are some special visualization methods for NNs. Duch [27] proposed an approach for visualization of NNs applied on classification problems. His method can be applied for problems with K classes if the output is coded as a K-length vector: $(1, 0, \ldots, 0)$ means the first class, $(0, 1, \ldots, 0)$ the second, and so on. In this case the classes are represented by the corners of the K-dimensional unit hypercube. The approach proposed by Duch maps the NN *output* into two dimensions; it basically 'flattens' the hypercube into two dimensions. This approach was thought over and applied on the *output of the hidden neurons* z_j in [28, 29]. The method was more straightforward than the former one because the hidden variables (the activation functions) take values from $[0, 1]$, and therefore the h-dimensional vectors are located within the unit hypercube. This method can be used not only for classification, but also for function approximation. Based on this latter approach, a picture of the behavior of the hidden units, their firing strength and their acti-

vation or saturation level can be obtained. The main drawback is that the number of classes/hidden neurons is limited. To keep the figures simple and interpretable, only three to six variables can be used. In the following, a different method is proposed to visualize the NN. The presented approach utilizes the *antecedent part* of the extracted fuzzy rules (since OLS based model reduction uses the consequent parts; see Sect. 3.3). Reducing the FAS rule base by analyzing the antecedent part of the rules is possible by measuring the similarity of the membership functions, and removing the too-similar neurons. Utilizing the equality of FAS and NN, the following classical interclass separability measure (originally for fuzzy systems) could be used to compare the univariate functions decomposed from hidden neurons:

$$S^i_{j,l,k} = \frac{\int \min\left(A^i_{j,k}(x_i), A^i_{l,k}(x_i)\right) dx_i}{\int \max\left(A^i_{j,k}(x_i), A^i_{l,k}(x_i)\right) dx_i}, \tag{3.11}$$

where $i = 1, \ldots, n,\ \ j, l = 1, \ldots, h$. Equation 3.11 can be used to measure the similarity of two *clauses* in the rule base, in other words, the similarity of two hidden neurons for the same input variables (see Fig. 3.6). To compare the hidden neurons themselves with multivariate activation functions, the following measure seems to be straightforward:

$$S_{j,l,k} = \prod_i S^i_{j,l,k},\ \ i = 1, \ldots, n,\ \ j, l = 1, \ldots, h. \tag{3.12}$$

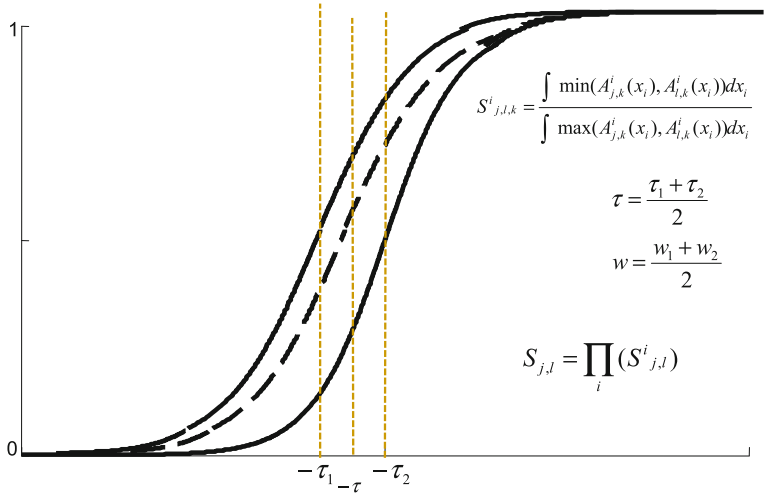

Fig. 3.6 Similarity of membership functions

With this measure, pairwise similarities of hidden neurons in the range of [0, 1] can be obtained. To get pairwise distances if needed, the simple form of $1 - S_{j,l,k}$ can be used. Based on these distances, which can be called relative data, the neurons themselves can be mapped onto two dimensions. In this book, classical multidimensional scaling is used.

Multidimensional scaling (MDS) refers to a group of unsupervised data visualisation techniques. Given a set of data in a high-dimensional feature space, MDS maps them into a low-dimensional (generally two-dimensional) data space in a way that objects that are very similar to each other in the original space are placed near each other on the map, and objects that are very different from each other are placed far away from each other. There are two types of MDS: (i) *metric MDS* and (ii) *non-metric MDS*.

Metric (or classical) MDS discovers the underlying structure of a data set by preserving similarity information (pairwise distances) among the data objects. Similarly to the Sammon mapping, the metric multidimensional scaling also tries to minimise a stress function. If the square-error cost is used, the objective function (stress) to be minimized can be written as

$$E_{metric\_MDS} = \frac{1}{\sum_{i<j}^{N} d_{i,j}^{*2}} \sum_{i<j}^{N} (d_{i,j}^{*} - d_{i,j})^2, \tag{3.13}$$

where $d_{i,j}^{*}$ denotes the distance between the vectors $\mathbf{x}_i$ and $\mathbf{x}_j$, and $d_{i,j}$ between $\mathbf{y}_i$ and $\mathbf{y}_j$. The only difference between the stress functions of the Sammon mapping and the metric MDS (see 3.13) is that the errors in distance preservation in the case of Sammon mapping are normalized by the distances of the input data objects. Because of this normalisation the Sammon mapping emphasises the preservation of small distances.

Classical MDS is an algebraic method based on the fact that matrix $\mathbf{Y}$ containing the output coordinates can be derived by eigenvalue decomposition from the scalar product matrix $\mathbf{B} = \mathbf{Y}\mathbf{Y}^T$. Matrix $\mathbf{B}$ can be found from the known distances using the Young-Householder process. The detailed metric MDS algorithm is the following

Metric MDS Algorithm

1. Let the searched coordinates of n points in a d-dimensional Euclidean space be given by $\mathbf{y}_i$ $(i = 1, \ldots, n)$, where $\mathbf{y}_i = \left[y_{i,1}, \ldots, y_{i,d}\right]^T$. Matrix $\mathbf{Y} = \left[\mathbf{y}_1, \ldots, \mathbf{y}_n\right]^T$ is the $n \times d$ coordinate matrix. The Euclidean distances $\{d_{i,j} = \left(\mathbf{y}_i - \mathbf{y}_j\right)^T \left(\mathbf{y}_i - \mathbf{y}_j\right)\}$ are known. The inner product of matrix $\mathbf{Y}$ is denoted by $\mathbf{B} = \mathbf{Y}\mathbf{Y}^T$. Find matrix $\mathbf{B}$ from the known distances $\{d_{i,j}\}$ using the Young-Householder process:
   (a) Define matrix $\mathbf{A} = [a_{i,j}]$, where $a_{i,j} = -\frac{1}{2} d_{i,j}^2$,
   (b) Deduce matrix $\mathbf{B}$ from $\mathbf{B} = \mathbf{HAH}$, where $\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{l}\mathbf{l}^T$ is the centering matrix, and $\mathbf{l}$ is an $n \times 1$ column vector of n 1s.

2. Recover the coordinate matrix $\mathbf{Y}$ from $\mathbf{B}$ using the spectral decomposition of $\mathbf{B}$:

   (a) The inner product matrix $\mathbf{B}$ is expressed as $\mathbf{B} = \mathbf{Y}\mathbf{Y}^T$. The rank of $\mathbf{B}$ is $r(\mathbf{B}) = r\left(\mathbf{Y}\mathbf{Y}^T\right) = r(\mathbf{Y}) = d$. $\mathbf{B}$ is symmetric, positive semi-definite and of rank d, and hence has d nonnegative eigenvalues and $n - d$ zero eigenvalues.

   (b) Matrix $\mathbf{B}$ is now written in terms of its spectral decomposition, $\mathbf{B} = \mathbf{V}\Lambda\mathbf{V}^T$, where $\Lambda = diag[\lambda_1, \lambda_2, \ldots, \lambda_n]$, the diagonal matrix of eigenvalues λ_i of $\mathbf{B}$, and $\mathbf{V} = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$, the matrix of corresponding eigenvectors, normalized such that $\mathbf{v}_i^T\mathbf{v}_i = \mathbf{l}$,

   (c) Because of the $n - d$ zero eigenvalues, $\mathbf{B}$ can now be rewritten as $\mathbf{B} = \mathbf{V}_1\Lambda_1\mathbf{V}_1^T$, where $\Lambda_1 = diag[\lambda_1, \lambda_2, \ldots, \lambda_d]$ and $\mathbf{V}_1 = [\mathbf{v}_1, \ldots, \mathbf{v}_d]$,

   (d) Finally the coordinate matrix is given by $\mathbf{Y} = \mathbf{V}_1\Lambda_1^{\frac{1}{2}}$, where $\Lambda_1^{\frac{1}{2}} = diag\left[\lambda_1^{\frac{1}{2}}, \ldots, \lambda_d^{\frac{1}{2}}\right]$.

In contrast with metric multidimensional scaling, in *non-metric MDS* only the ordinal information of the proximities is used for constructing the spatial configuration. Thereby non-metric MDS attempts to preserve the rank order among the dissimilarities. The non-metric MDS finds a configuration of points whose pairwise Euclidean distances have approximately the same rank order as the corresponding dissimilarities of the objects. Equivalently, the non-metric MDS finds a configuration of points whose pairwise Euclidean distances approximate a monotonic transformation of the dissimilarities. These transformed values are known as the disparities. The non-metric MDS stress can be formulated as follows:

$$E_{nonmetric\_MDS} = \sqrt{\sum_{i<j}^{N}(\widehat{d}_{i,j} - d_{i,j})^2 / \sum_{i<j}^{N} d_{i,j}^2}, \tag{3.14}$$

where $\widehat{d}_{i,j}$ yields the disparity of objects $\mathbf{x}_i$ and $\mathbf{x}_j$, and $d_{i,j}$ denotes the distance between the vectors $\mathbf{y}_i$ and $\mathbf{y}_j$. Traditionally, the non-metric MDS stress is often called Stress-1 due to Kruskal.

The main steps of the non-metric MDS algorithm are given in Algorithm 1.

Algorithm 1 Non-metric MDS algorithm

Step 1 Find a random configuration of points in the output space.
Step 2 Calculate the distances between the points.
Step 3 Find the optimal monotonic transformation of the proximities in order to obtain the disparities.
Step 4 Minimize the non-metric MDS stress function by finding a new configuration of points.
Step 5 Compare the stress to some criteria. If the stress is not small enough then go back to Step 2.

It can be shown that metric and non-metric MDS mappings are substantially different methods. On the one hand, the while metric MDS algorithm is an algebraic method, the non-metric MDS is an iterative mapping process. On the other hand, the main goal of the optimisation differs significantly, too. While metric multidimen-

sional scaling methods attempt to maintain the degree of the pairwise dissimilarities of data points, the non-metric multidimensional scaling methods focus on the preservation of the order of the neighborhood relations of the objects.

The mapped two dimensional points refer to how similarly the neurons behave. As can be seen, the above-mentioned approaches [28, 29] visualize the output of the hidden neurons, and draw conclusions from the location of these data. The proposed approach focuses on the behavior of hidden neurons as well, but utilizes the shape of the identified multidimensional activation functions. The previous approach can be used to determine how well the NN was trained, since the proposed one shows which neurons are similar and redundant within the trained network. In this formulation, this method can instead be used for complexity reduction purposes, and not to qualify the training procedure.

3.5 Application Example

For applying the introduced visualization and reduction techniques, we used a dataset of a pH process (see [32] or Appendix C), where the concentration of hydrogen ions in a continuously stirred tank reactor is modeled (CSTR). This well-known modeling problem presents difficulties due to the nonlinearity of the process dynamics. This process can be correctly modeled as a first-order input-output (NARX) system, where the actual output (the pH), $y(k+1)$, depends on the pH of the reactor, $y(k)$, and the NaOH feed, $u(k)$, at the kth sample time (sample time is $t_s = 0.2$ min):

$$y(k+1) = f\left(y\left(k\right), u\left(k\right)\right). \tag{3.15}$$

Parameters of the neural network were identified by the backpropagation algorithm based on uniformly distributed training data where F_{NaOH} is in the range 515–525 l/min. Our experiences show that seven neurons are sufficient in the hidden layer of the NN. The results in Table 3.1 show that the neural network models give very good prediction performance for this process. The numbers in brackets represent the number of neurons in the hidden layer and the neurons removed from the identified NN.

Applying the proposed visualization and transformation techniques, Fig. 3.7 shows the decomposed univariate membership functions; and the two-dimension mapped distance matrix according to the NN model parameters can be seen in Fig. 3.8. For better interpretability, the histograms of the corresponding model inputs are illustrated in the last two subplots.

In Fig. 3.8 the pairwise distances of the neurons (see Eq. 3.11 in the previous section) were mapped into two dimensions with MDS, the and two-dimensional points refer to how similarly the neurons behave.

The neurons are listed according to the OLS ranking on the left of Fig. 3.9, starting with the rules decomposed from the most important neuron in the hidden network layer. The consequence of synthesizing the results is that it is possible to remove

Table 3.1 One-step ahead prediction results

Testcase	Training errors (MSE)	Testing errors (MSE)
Neural Network (7)	3.088e-005	3.267e-005
Using i-or (7)	3.053e-005	3.259e-005
Network reduction (8/1 neuron)	4.434e-005	4.285e-005
Network reduction (7/1 neuron)	3.060e-005	3.247e-005
Network reduction (6/2 neuron)	2.884e-004	3.690e-004
Network reduction (6/1 neuron)	1.086e-004	1.316e-004

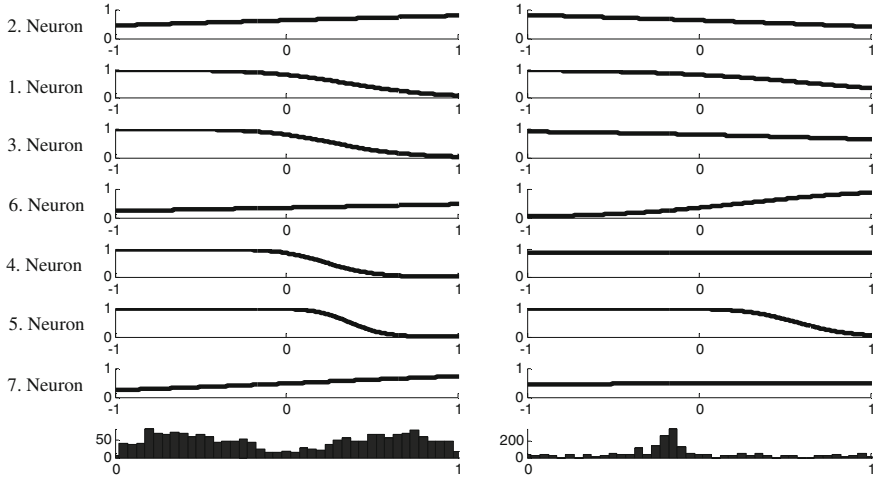

Fig. 3.7 Decomposed univariate membership functions

one neuron out of seven (in FAS the corresponding rules) from the model without a significant increase in modeling performance, because of the low error reduction rate of the last, seventh neuron. This achievement harmonizes with the issues of the mapped distances, where the second and the seventh neurons are closer to each other, but OLS-based ranking indicates the second one as more important.

Model reduction and visualization techniques such as OLS makes it possible to overcome the problem of overfitting and the performance of the reduced model is almost the same as that of the original one. A rigorous test of NARX models is free run simulation, because the errors can be cumulated. The result indicates the goodness of the reduced model even by free run simulation (3.5×10^{-3} for a neural

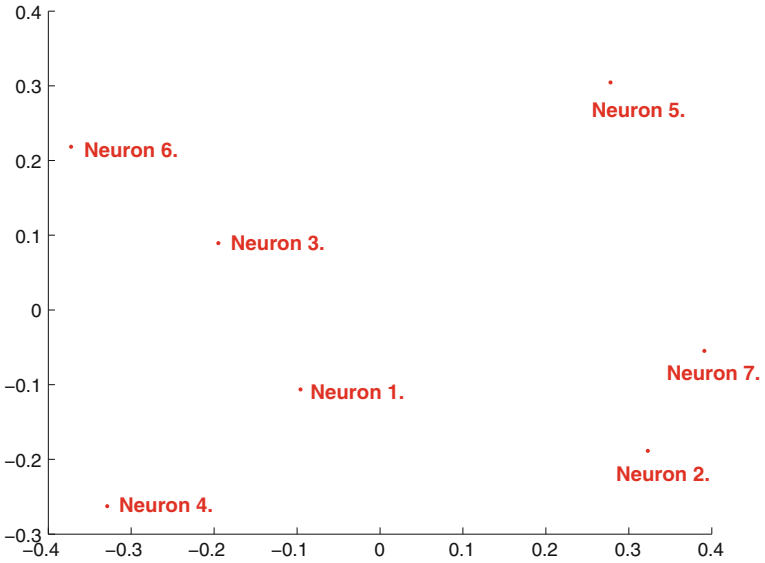

Fig. 3.8 Distances between neurons mapped into two dimensions with MDS

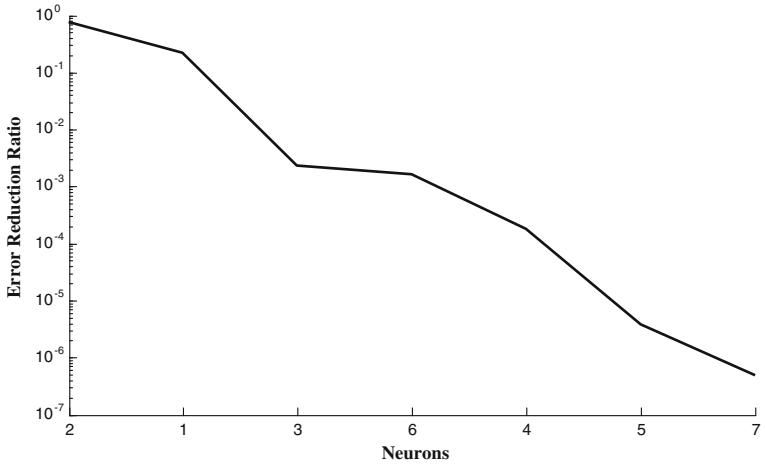

Fig. 3.9 Error reduction ratios

network with seven neurons, 3.823×10^{-3} using i-or for FAS with seven rules, 3.824×10^{-3} after removing one neuron from the hidden layer containing seven neurons).

Table 3.2 shows the training errors for neural networks with different numbers of neurons in the hidden layer (six to 15). These models were reduced by one to 14 neurons. The obtained results indicate that it is worth considering selecting the

Table 3.2 Training errors for different model structures and model reductions

–	6	7	8	9	10	11	12	13	14	15
1	6.8×10^{-4}	4.4×10^{-5}	2.7×10^{-5}	3.4×10^{-5}	4.5×10^{-5}	2.9×10^{-5}	3.3×10^{-5}	6.6×10^{-4}	3.6×10^{-5}	6.6×10^{-4}
2	4.8×10^{-5}	6.7×10^{-4}	3.9×10^{-5}	3.9×10^{-5}	6.7×10^{-4}	3.4×10^{-5}	3.3×10^{-5}	3.6×10^{-5}	5.9×10^{-5}	1.1×10^{-4}
3	6.5×10^{-5}	3.8×10^{-5}	1.7×10^{-5}	1.4×10^{-4}	3.1×10^{-5}	2.7×10^{-5}	8.9×10^{-5}	3.3×10^{-5}	3.5×10^{-5}	3.8×10^{-5}
4	6.4×10^{-5}	3.3×10^{-5}	3.6×10^{-5}	5.0×10^{-5}	3.2×10^{-5}	6.6×10^{-4}	5.8×10^{-5}	3.9×10^{-5}	3.3×10^{-5}	3.8×10^{-5}
5	7.7×10^{-5}	2.3×10^{-5}	5.3×10^{-5}	3.4×10^{-5}	3.5×10^{-5}	3.8×10^{-5}	3.4×10^{-5}	3.8×10^{-5}	3.5×10^{-5}	4.2×10^{-5}
6	–	2.9×10^{-5}	4.4×10^{-5}	5.6×10^{-5}	5.2×10^{-5}	6.3×10^{-4}	6.5×10^{-4}	3.1×10^{-5}	8.4×10^{-5}	7.2×10^{-5}
7	–	–	6.4×10^{-4}	4.4×10^{-5}	3.5×10^{-5}	6.2×10^{-5}	4.1×10^{-5}	5.2×10^{-5}	3.4×10^{-5}	4.4×10^{-5}
8	–	–	–	1.7×10^{-5}	6.1×10^{-5}	7.9×10^{-5}	3.7×10^{-5}	3.2×10^{-5}	3.6×10^{-5}	7.1×10^{-5}
9	–	–	–	–	4.8×10^{-5}	4.4×10^{-5}	1.7×10^{-5}	4.9×10^{-5}	3.7×10^{-5}	4.7×10^{-5}
10	–	–	–	–	–	3.2×10^{-5}	3.2×10^{-5}	8.7×10^{-5}	9.0×10^{-5}	1.2×10^{-4}
11	–	–	–	–	–	–	3.1×10^{-5}	7.0×10^{-5}	5.0×10^{-5}	1.5×10^{-4}
12	–	–	–	–	–	–	–	3.6×10^{-5}	3.3×10^{-5}	4.5×10^{-5}
13	–	–	–	–	–	–	–	–	3.1×10^{-5}	1.2×10^{-4}
14	–	–	–	–	–	–	–	–	–	2.6×10^{-5}

Number of neurons in the hidden layer of the network/Number of reduced neurons from the given model structure

appropriate model structure, because better modeling performance can be achieved for reducing an overfitted model. The reason for this phenomenon is that the gradient-based training algorithms may stop in different local minimums.

3.6 Conclusions

Neural networks are often too complex and not interpretable; therefore, it is very difficult to utilize them correctly. This book proposed a new complex approach for visualization and reduction of neural networks, and argued that neural networks with sigmoid transfer functions are identical to fuzzy additive systems.

The used similarity measure can be applied for further reduction of the rule base. This can be done in an automatic way if a threshold value is defined previously. If the measured similarity is greater than the threshold, the corresponding two neurons in the original neural network can be considered as identical; therefore, further reduction of the FAS rule base is possible. This technique can be used even during the learning process of the neural network.

A possible future research direction is to develop a new learning procedure for neural networks using prior knowledge-based if-then rules, which combines the user's experience and/or constraints with the learning capability of a NN.

Chapter 4
Interpretability of Support Vector Machines

Application of support vector methods for the initialization of fuzzy models is not a completely new idea. Numerous methods have been proposed to build the connection between the SVR and the FIS. Chen and Wang [22, 23] propose a positive definite fuzzy system (PDFS). In the proposed fuzzy model, the PDFS is equivalent to a Gaussian kernel SVM [22] if Gaussian membership functions are adopted. The antecedent of a fuzzy rule is obtained by a support vector (SV). Therefore the number of fuzzy rules is the same as the number of SVs. As the number of SVs is generally large, the size of the FIS based on an SVM is also large. To solve this problem, researchers [79] proposed a learning algorithm to remove the irrelevant fuzzy rules. In spite of this, the generalization performance is degraded. The above methods are for a zero-order FIS, which has one fuzzy singleton in the consequent of a fuzzy rule. For the first order FIS, Leski [25] describes a method for obtaining one by means of the SVM with data-independent kernel matrix. Moreover, Juang et al. used a combination of fuzzy clustering and the linear SVM to establish a fuzzy model with fewer parameters and better generalization performance. However, negligible effort has been made to establish a HFIS (high order-FIS) with kernel methods. In [80] the authors presented an HFIS with high accuracy and good generalization performance. It was shown how to obtain the formulation of the nonlinear function for the consequent part.

Furthermore, Catala used prototype vectors to combine with the support vectors using geometric methods to define ellipsoids in the input space, which are later transformed to if-then rules [21]. In [81], a special operator was utilized to achieve equivalency between support vector machines and fuzzy rule-based systems. In [82], the utilization of support vector models is described to solve the convex optimization problem for multivariate linear regression models, and it is also shown how a multivariate fuzzy nonlinear regression model can be formalized for numerical inputs and fuzzy output. Multiple types of kernels [81, 82] can be used to solve crisp nonlinear regression problems [83]. Chia et al. [84] used a combination of fuzzy clustering and linear support vector regression to obtain Takagi-Sugeno-type fuzzy rules. Support vector machines can be applied to determine the support vectors for each fuzzy cluster

T. Kenesei and J. Abonyi, *Interpretability of Computational Intelligence-Based Regression Models*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-21942-4_4

obtained by the fuzzy c-means clustering algorithm [85]. The visualization of fuzzy regression models was also discussed previously. The interpretation of fuzzy regression is provided with an insight into regression intervals so that regression interval analysis, data type analysis and variable selection are analytically performed [86]. In [87], a visualization and interpretation tool is presented.

Feature space is visualized by highlighting the corresponding variables in the original input data to show how they are associated to the output variable. This shows which part of the input data can be utilized to estimate the output value. This technique also describes which input variables are responsible for the performance of the support vector regression. With the combination of visualization and interpretation, the black-box support vector regression is identified in one step.

It must be taken into account that fuzzy logic does not guarantee interpretability as a prerequisite, because obtaining fuzzy models from support vector-based training often results in fuzzy models with a high number of fuzzy rules. This phenomenon makes interpretability much more difficult; therefore, the aim of this chapter is to describe a combination-of-tools three-step technique on how to use reduction techniques on trained SVR models to acquire transparent but accurate fuzzy rule-based regression models. The steps are the following:

1. *Application of the Reduced Set Method*
   The identification of the SVM is followed by the application of the Reduced Set (RS) method to decrease the number of kernel functions. Originally, this method was introduced by [88] to reduce the computational complexity of SVMs. The obtained SVM is subsequently transformed into a fuzzy rule-based regression model.
2. *Similarity-Based Fuzzy Set Merging*
   The Gaussian membership functions of the fuzzy rule-based regression model are derived from the Gaussian kernel functions of the SVM. The interpretability of a fuzzy model highly depends on the distribution of the membership functions. Hence, the next reduction step is achieved by merging fuzzy sets based on a similarity measure [89].
3. *Rule-Based Simplification by Orthogonal Transformation*
   Finally, an orthogonal least squares method is used to reduce the number of rules and reestimate the consequent parameters of the regression model. The application of orthogonal transformations for reducing the number of rules has received much attention in the recent literature [90, 91]. These methods evaluate the output contribution of the rules to obtain the order of importance. The less important rules are then removed according to this ranking to further reduce the complexity and increase the transparency.

This chapter is organized as follows. Firstly basic notions of support vector machines and their connection with fuzzy regression are described. After a detailed description of the three-step reduction algorithm, examples indicating the power and the usage of the described techniques on regression problems are presented.

4.1 FIS-Interpreted SVR

SVM has been recently introduced for solving pattern recognition and function estimation problems. SVM is a nonlinear generalization of the Generalized Portrait algorithm developed in Russia in the 1960s. In its present form, SVM was developed at AT&T Bell Laboratories by Vapnik and coworkers [24]. SVM learning has now evolved into an active area of research. Moreover, the technique belongs to the standard method toolbox of machine learning.

4.1.1 Support Vector Regression Models

The basic idea behind support vector regression is the kernel function: $k\left(\mathbf{x}_i, \mathbf{x}_j\right)$. Using k instead of dot product in $\mathbf{R}_i^N$ corresponds to mapping the data into a possibly high-dimensional space F by a usually nonlinear map $\phi : \mathbf{R}_i^N \rightarrow F$ and taking the dot product there:

$$k\left(\mathbf{z}_i, \mathbf{x}\right) = \left(\phi\left(\mathbf{z}_i\right), \phi\left(\mathbf{x}\right)\right). \tag{4.1}$$

The SVR concept will be introduced based on [92]; for more details see [17]. Suppose we have training data $\left\{\left(x_1, y_1\right), \ldots, \left(x_{N_d}, y_{N_d}\right)\right\} \subset \chi \times \mathbf{R}_i^N$, where χ denotes the space of input patterns. The aim is to find a function $f\left(x\right)$ that has at most ε deviation from the targets with the obtained y_i for all the training data. In other words, we do not care about the errors as long as they are less than ε, but any larger deviation than ε won't be accepted. SVR can be formulated as follows:

$$\begin{aligned} \min_{w,b,\xi_i,\xi_i} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N_d}\left(\xi_i + \xi_i^*\right) \\ s.t.\ \ y_i - \mathbf{w}^T\phi\left(\mathbf{x}_i\right) - b \leq \varepsilon + \xi_i, \\ \mathbf{w}^T\phi\left(\mathbf{x}_i\right) + b \leq \varepsilon + \xi_i^* - y_i, \\ \xi_i, \xi_i^* \geq 0, \end{aligned} \tag{4.2}$$

where ϕ is the feature mapping for kernel k, ε is the tolerance error, ξ_i, ξ_i^* are slack variables and $C > 0$ is a cost coefficient, which determines the trade-off between the model complexity and the degree of tolerance to the errors larger than ε. The dual form of the optimization problem (4.2) becomes a quadratic programming (QP) problem:

$$\max_{\alpha,\alpha^*} -\frac{1}{2}\sum_{i,j=1}^{N_d}\left(\alpha_i-\alpha_i^*\right)\left(\alpha_j-\alpha_j^*\right)k\left(\mathbf{x}_i,\mathbf{x}_j\right) - \varepsilon\sum_{i=1}^{N_d}\left(\alpha_i+\alpha_i^*\right)+\sum_{i=1}^{N_d} y_i\left(\alpha_i-\alpha_i^*\right)$$
$$s.t.\ \sum_{i=1}^{N_d} y_i\left(\alpha_i-\alpha_i^*\right)=0\ \ \alpha,\alpha^*\in[0,C]\,, \tag{4.3}$$

where α and α^* are the Lagrange multipliers. As an outcome of solving the QP problem, Eq. 4.3 can be rewritten to the following form:

$$f\left(x\right)=\sum_{i=1}^{N_d}\left(\alpha_i-\alpha_i^*\right)k\left(\mathbf{x}_i,\mathbf{x}\right)+b. \tag{4.4}$$

Let $\gamma_i=\alpha_i-\alpha_i^*$. In case $\gamma_i\neq 0$, the corresponding training pattern x_i can be noted as a support vector.

4.1.2 Structure of Fuzzy Rule-Based Regression Model

To get a fuzzy rule-based regression model from the support vector regression model, the following interpretation is needed:

$$y=\sum_{i=1}^{N_R}\beta_i(\mathbf{x})\delta_i+b, \tag{4.5}$$

where β_i is the firing strength and δ_i is the rule consequent. The output of the regression model is calculated by this equation. In the case of fuzzy systems, fuzzy rules can be formulated as follows

$$R_i\ if\ \ x_1\ is\ A_{i1}\ and\ldots x_n\ is\ A_{in}\ then\ y_i=\delta_i,\quad i=1,\ldots,N_R\,, \tag{4.6}$$

where R_i is the ith rule in the fuzzy rule-based regressor and N_R denotes the number of rules. $A_i,\ldots,A_{N_i}$ denote the antecedent fuzzy sets that define the operating region of the rule in the N_i-dimensional input space. The rule consequent δ_i is a crisp number. The connective is modeled by the product operator. Hence the degree of activation of the ith rule is calculated as

$$\beta_i(\mathbf{x})=\prod_{j=1}^{N_i}A_{ij}(\mathbf{x}_j),\quad i=1,\ldots,N_R. \tag{4.7}$$

The main principle of kernel-based support vector regressors is the identification of a linear decision boundary in this high-dimensional feature space. The link to the fuzzy model structure is the following: The fuzzy sets are represented by Gaussian membership functions

$$A_{ij}(\mathbf{x}_j) = exp\left(\frac{\left(\mathbf{x}_j - \mathbf{z}_{ij}\right)^2}{2\sigma^2}\right). \tag{4.8}$$

The degree of fulfilment $\beta_i(\mathbf{x})$ can be written through Eqs. 4.7 and 4.8 in a more compact form by using Gaussian kernels:

$$\beta_i(\mathbf{x}) = exp\left(\frac{\|\mathbf{x} - \mathbf{z}_j\|^2}{2\sigma^2}\right). \tag{4.9}$$

This kernel interpretation of fuzzy systems shows that fuzzy models are effective in solving nonlinear problems because they map the original input space into a nonlinear feature space by using membership functions similarly to the support vector machine that utilize kernel functions for this purpose.

4.2 Ensuring Interpretability with a Three-Step Algorithm

In the previous sections it was shown how an SVM that is structurally equivalent to a fuzzy model can be identified. Unfortunately, this identification method cannot be used directly for the identification of interpretable fuzzy systems because the number of the support vectors is usually very large. Typical values are 40–60 % of the training data, which is in our approach equal to the number of rules in the fuzzy system. Therefore, there is a need for an interpretable approximation of the support vector expansion. For this purpose, a stepwise algorithm will be introduced, where the first step is based on the recently published Reduced Set (RS) method developed for reducing the computational demand for the evaluation of SVMs [88].

4.2.1 Model Simplification by Reduced Set Method

The aim of the RS method is to approximate the high-dimensional feature space given by the support vectors

$$\Psi = \sum_{i=1}^{N_x} \gamma_i \phi(\mathbf{x}_i) \tag{4.10}$$

by a reduced set expansion

$$\Psi' = \sum_{i=1}^{N_R} \delta_i \phi(\mathbf{z}_i) \ , \tag{4.11}$$

with $N_R < N_x < N_d$, where N_x denotes the number of support vectors (the number of $\mathbf{x}_i$ vectors for those $\gamma_i \neq 0$), N_R represents the number of the desired rules in the fuzzy rule-based regressor that we would like to identify and $\mathbf{z}_i$ denotes the centers of the new kernel functions that are not necessarily training samples. N_R should be as small as possible because it determines the number of fuzzy rules. In practice it turns out that the RS method is often able to deliver a one-tenth reduction, so N_R can be chosen as $N_R = N_x/10$. For this model reduction, the squared error $\|\Psi - \Psi'\|^2$ has to be minimized. For this purpose, the 'kernel trick' has to be applied because ϕ is not given explicitly:

$$\begin{aligned} \|\Psi - \Psi'\|^2 \ &= \sum_{i,j=1}^{N_x} \gamma_i \gamma_j k(\mathbf{x}_i, \mathbf{x}_j) \\ &\quad + \sum_{i,j=1}^{N_R} \delta_i \delta_j k(\mathbf{z}_i, \mathbf{z}_j) - 2 \sum_{i}^{N_x} \sum_{j}^{N_R} \gamma_i \delta_j k(\mathbf{x}_i, \mathbf{z}_j) \ . \end{aligned} \tag{4.12}$$

The cost function Eq. 4.12 is minimized in a stepwise manner while the feature space is approximated by the following iterative algorithm:

Repeat for $m : 2, \ldots, N_R$;

- **Step 1**: *Obtain the residual space*

  Let Ψ_m be the residual of the feature space approximation generated at the $(m-1)$th step:

$$\begin{aligned} \Psi_m &= \sum_{i=1}^{N_x} \gamma_i \phi(\mathbf{x}_i) - \sum_{i=1}^{m-1} \delta_i \phi(\mathbf{z}_i) \\ &= \sum_{i=1}^{N_m} \varepsilon_i \phi(\mathbf{v}_i) \ , \end{aligned} \tag{4.13}$$

  where

$$\begin{aligned} &(\varepsilon_1, \ldots, \varepsilon_{N_m}) = (\gamma_1, \ldots, \gamma_{N_x}, -\delta_1, \ldots, -\delta_{m-1}) \ , \\ &(\mathbf{v}_1, \ldots, \mathbf{v}_{N_m}) = (\mathbf{x}_1, \ldots, \mathbf{x}_{N_x}, \mathbf{z}_1, \ldots, \mathbf{z}_{m-1}) \ , \\ &N_m = N_x + m - 1 \ . \end{aligned}$$

- **Step 2**: *Inner iteration step for determining* $\mathbf{z}_m$
  This residual function is approximated by the determination of $\mathbf{z}_m$ and δ_m in the iterative procedure, where the following cost function has to be minimized:

$$\min_{\delta_m, \mathbf{z}_m} \|\Psi_m - \delta_m \phi(\mathbf{z}_m)\|^2 . \tag{4.14}$$

  This can be done by standard techniques or using fixed–point iteration, as shown in [88]:

$$\mathbf{z}_m^{n+1} = \frac{\sum_{i=1}^{N_m} \varepsilon_i \exp\left(-\|\mathbf{v}_i - \mathbf{z}_m^n\|^2 / \left(2\sigma^2\right)\right) \mathbf{v}_i}{\sum_{i=1}^{N_m} \varepsilon_i \exp\left(-\|\mathbf{v}_i - \mathbf{z}_m^n\|^2 / \left(2\sigma^2\right)\right)} , \tag{4.15}$$

  where the superscript n denotes the nth inner iteration step; (4.15) is iterated till it converges to $\|\mathbf{z}_m^{n+1} - \mathbf{z}_m^n\|^2 < \varepsilon$.
  Interestingly, (4.15) can be interpreted in the context of clustering. It determines the center of a single Gaussian cluster, trying to capture as many of the $\mathbf{v}_i$ with positive δ_i as possible, and simultaneously avoiding those $\mathbf{v}_i$ with negative δ_i.
- **Step 3**: *Least squares estimation of the* δ_i *coefficients*
  The δ_m coefficient is calculated by recalculating the whole $\delta = [\delta_1, \ldots, \delta_m]^T$ vector by minimizing Eq. 4.12:

$$\delta = (K^z)^{-1} K^{zx} \gamma, \tag{4.16}$$

  where the element of the matrices are expressed by the kernel functions $K_{ij}^z = k(\mathbf{z}_i, \mathbf{z}_j)$ and $K_{ij}^{zx} = k(\mathbf{z}_i, \mathbf{x}_j)$.

4.2.2 Reducing the Number of Fuzzy Sets

In the previous section, it was shown how a kernel-based regression model with a given number of kernel functions, N_R, can be obtained. Because the number of rules in the transformed fuzzy system is identical to the number of kernels, it is extremely important to get a moderate number of kernels in order to obtain a compact fuzzy rule-based regression model.

From Eq. 4.9 it can be seen that the number of fuzzy sets in the identified model is $N_s = N_R N_i$. The interpretability of a fuzzy model highly depends on the distribution of these membership functions. With the simple use of Eq. 4.8, some of the membership functions may appear almost undistinguishable. Merging similar fuzzy sets reduces the number of linguistic terms used in the model and thereby increases model transparency. This reduction is achieved by a rule-base simplification method [89, 93], based on a similarity measure, $S(A_{ij}, A_{kj})$, $i, k = 1, \ldots, n$ and $i \neq j$.

If $S(A_{ij}, A_{kj}) = 1$, then the two membership functions A_{ij} and A_{kj} are equal. $S(A_{ij}, A_{kj})$ becomes 0 when the membership functions are nonoverlapping. During the rule-based simplification procedure, similar fuzzy sets are merged when their similarity exceeds a user-defined threshold $\theta \in [0, 1]$. The set similarity measure can be based on the set-theoretic operations of intersection and union [89],

$$S(A_{ij}, A_{kj}) = \frac{|A_{ij} \cap A_{kj}|}{|A_{ij} \cup A_{kj}|}, \tag{4.17}$$

where $|.|$ denotes the cardinality of a set, and the $\cap$ and $\cup$ operators represent the intersection and union, respectively; or it can be based on the distance of the two fuzzy sets. Here, the following expression was used to approximate the similarity between two Gaussian fuzzy sets [93]:

$$\begin{aligned} S(A_{ij}, A_{kj}) &= \frac{1}{1 + d(A_{ij}, A_{kj})} \\ &= \frac{1}{1 + \sqrt{(z_{ij} - z_{kj})^2 + (\sigma_{ij} - \sigma_{kj})^2}}. \end{aligned} \tag{4.18}$$

4.2.3 Reducing the Number of Rules by Orthogonal Transforms

By using the previously presented SVM identification and reduction techniques, the following fuzzy rule-based regression model has been identified:

$$y = \sum_{i=1}^{N_R} \prod_{j=1}^{N_i} \exp\left(\frac{\left(x_j - z_{ij}\right)^2}{2\sigma^2} \right) \delta_i + b. \tag{4.19}$$

Due to the applied RS method and the fuzzy set merging procedure, the obtained membership functions only approximate the original feature space identified by the SVM. Hence, the $\delta = [\delta_1, \ldots, \delta_{N_r}]^T$ consequent parameters of the rules have to be reidentified to minimize the difference between the decision function of the support vector machine, Eq. 4.4, and the fuzzy model Eq. 4.19:

$$MSE = \sum_{j=1}^{N_d} \left(\sum_{i=1}^{N_x} \gamma_i k(\mathbf{x}_j, \mathbf{x}_i) - \sum_{i=1}^{N_R} \delta_i \beta_i(\mathbf{x}_j) \right)^2 \tag{4.20}$$

$$= \|\mathbf{y}_s - \mathbf{B}\delta\|^2, \tag{4.21}$$

where the matrix $\mathbf{B} = [\mathbf{b}_1, \ldots, \mathbf{b}_{N_R}] \in \mathbf{R}^{N_d \times N_R}$ contains the firing strength of all N_R rules for all the inputs $\mathbf{x}_i$, where $\mathbf{b}_j = [\beta_j(\mathbf{x}_1), \ldots, \beta_j(\mathbf{x}_{N_d})]^T$. As the fuzzy

rule-based regression model (4.19) is linear in the parameters δ, Eq. 4.20 can be solved by a least squares method (see Appendix B and Eq. B.1 for further details).

The application of orthogonal transformations for the above-mentioned regression problem Eq. 4.20 for reducing the number of rules has received much attention in recent literature [90, 91].

For modeling purposes, the Orthogonal Least Squares (OLS) is the most appropriate tool [90]. The OLS method transforms the columns of $\mathbf{B}$ into a set of orthogonal basis vectors in order to inspect the individual contribution of each rule.

To do this, Gram-Schmidt orthogonalization of $\mathbf{B} = \mathbf{WA}$ is used, where $\mathbf{W}$ is an orthogonal matrix $\mathbf{W}^T\mathbf{W} = \mathbf{I}$ and $\mathbf{A}$ is an upper triangular matrix with unity diagonal elements. If $\mathbf{w}_i$ denotes the ith column of $\mathbf{W}$ and g_i is the corresponding element of the OLS solution vector $\mathbf{g} = \mathbf{A}\delta$, the output variance $\mathbf{y}_s^T\mathbf{y}_s/N_d$ can be explained by the regressors $\sum_{i=1}^{N_r} g_i\mathbf{w}_i^T\mathbf{w}_i/N_d$. Thus, the error reduction ratio, ρ, due to an individual rule i can be expressed as

$$\rho^i = \frac{g_i^2\mathbf{w}_i^T\mathbf{w}_i}{\mathbf{y}_s^T\mathbf{y}_s}. \tag{4.22}$$

This ratio offers a simple mean for ordering the rules, and can be easily used to select a subset of rules in a forward regression manner.

Evaluating only the approximation capabilities of the rules, the OLS method often assigns high importance to a set of redundant or correlated rules. To avoid this, in [91, 94] some extensions for the OLS method were proposed.

4.3 Application Examples

4.3.1 Illustrative Example

To demonstrate the potential of Support Vector Regression techniques two examples were introduced. Firstly, an illustrative regression problem was solved with a simple dataset containing 51 samples (Fig. 4.1). The SVR technique obtained 14 support vectors. This model has been reduced by the RS method (Step 1), by which we have tried to reduce the model to operate with ten rules. The modeling results can be seen in Table 4.1. The utilization of all the three algorithm steps reduced the number of fuzzy rules to six, however this indicated slight increase in modeling error.

4.3.2 Identification of Hammerstein System

In this example, the support vector regression is used to approximate a Hammerstein system that consists of a series connection of a memoryless nonlinearity, f, and

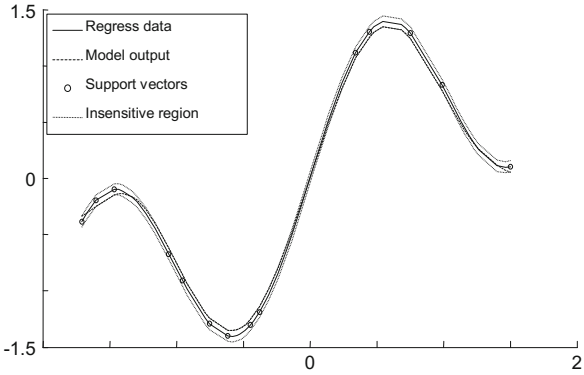

Fig. 4.1 Illustrative example with model output, support vectors and the insensitive region

Table 4.1 Results on regression data

Method	RMSE	#Rules
SVR Identification	0.084	14
Step 1 reduction	0.0919	10
Step 2 reduction	0.2415	9
Step 3 reduction	0.3361	6

linear dynamics, G, as shown in Fig. 4.2, where v represents the transformed input variable. For transparent representation, the Hammerstein system consists of a first-order linear part $y\,(k+1) = 0,9y(k) + 0,1v(k)$ and a static nonlinearity represented by a polynomial, $v(k) = u(k)^2$. The dataset contains 500 input-output data. The support vector regression model had the efficiency summarized in Table 4.2.

As Fig. 4.3 and Table 4.2 conclude, support vector regression is able to give accurate models for Hammerstein system identification. Extracted, nondistinguishable rules from this system are represented in Fig. 4.4, therefore the three-step reduction algorithm is used to acquire interpretable models.

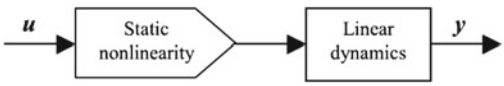

Fig. 4.2 Hammerstein system

Table 4.2 Results on Hammerstein system identification

Method	RMSE	#Rules
SVR identification	0.0533	22
Step 1 reduction	0.0604	15
Step 2 reduction	0.0650	13
Step 3 reduction	0.0792	12

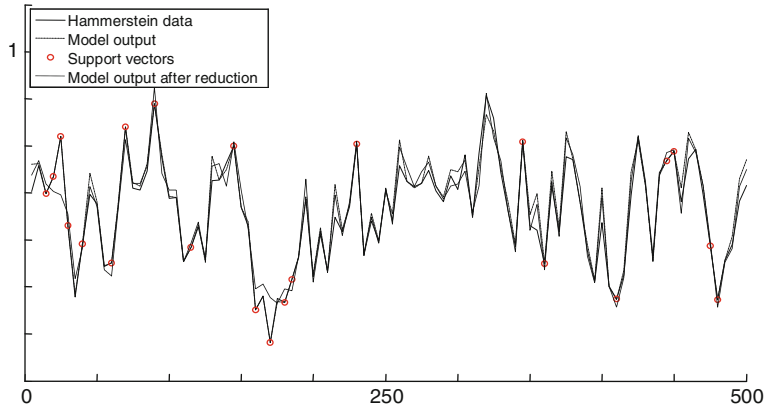

Fig. 4.3 Identified Hammerstein system, support vectors and model output after reduction

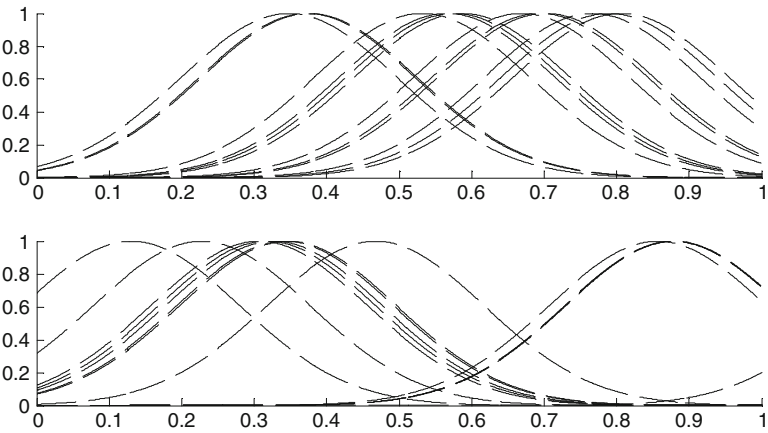

Fig. 4.4 Non-distinguishable membership functions obtained after the application of RS method

After applying the RS method (Step 1), the number of rules could be reduced to 15 without any major modeling error increase. Using further reductions with the second and third steps of the proposed algorithm, interpretable model (see Fig. 4.5) and an accurate model could be extracted.

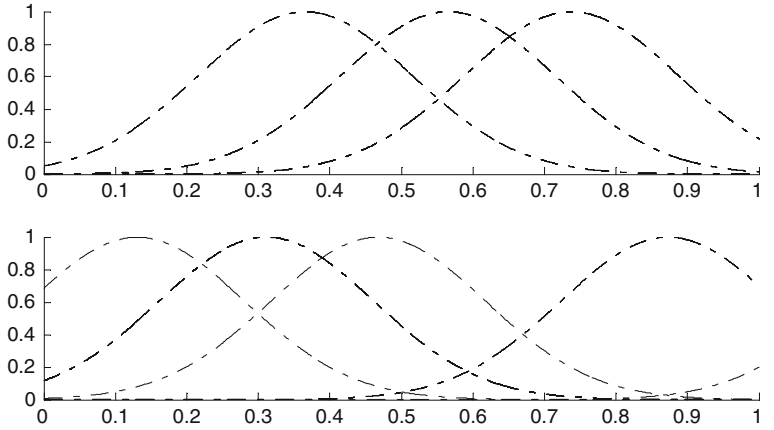

Fig. 4.5 Interpretable membership functions of the reduced fuzzy model

4.4 Conclusions

Support vector-based techniques and fuzzy rule-based models work in a similar manner as both models map the input space of the problem into a feature space with the use of either nonlinear kernel or membership functions. The main difference between support vector-based and fuzzy rule-based systems is that fuzzy systems have to fulfil two objectives simultaneously, i.e., they must provide a good modeling performance and must be linguistically interpretable, which is not an issue for support vector systems. However, as the structure identification of fuzzy systems is a challenging task, the application of kernel-based methods for model initialization could be advantageous because of the high performance and the good generalization properties of these type of models.

Accordingly, support vector-based initialization of fuzzy rule-based models is used. First, the initial fuzzy model is derived by means of the support vector learning algorithm. Then the support vector model is transformed into an initial fuzzy model that is subsequently reduced by means of the reduced set method, similarity-based fuzzy set merging, and orthogonal transform-based rule reduction. Because these rule-based simplification steps do not utilize any nonlinear optimization tools, it is computationally cheap and easy to implement them. The application of the proposed approach was shown on simple one–dimensional function identification data and Hammerstein system identification. The obtained models are very compact but their accuracy is adequate. Besides, it might be clear that real progress still can be made in the development of novel methods for feature selection.

Chapter 5
Summary

A majority of problems in engineering practice require data-driven modeling of nonlinear relationships between experimental and technological variables. The complexity of nonlinear regression techniques is gradually expanding with the development of analytical and experimental techniques; hence model structure and parameter identification is a current and important topic in the field of nonlinear regression, not just from a scientific but also from an industrial point of view as well. Model interpretability is the most important key property besides accuracy in the regression modeling of technological processes, and this is an essential characteristic of these models in their application as process controllers. As was mentioned above, model structure and parameter identification is a topic of increasing importance, since an identified model needs to be interpretable as well. In line with these expectations, and taking the interpretability of regression models as a basic requirement, robust nonlinear regression identification algorithms were developed in this book. Three algorithms were examined in detail, namely identification of regression tree based hinging hyperplanes, neural networks, and support vector regression. The application of these techniques eventuate in black-box models at the first step. We show in our book how interpretability could be maintained during model identification with utilization of applicable visualization and model structure reduction techniques within the fuzzy modeling framework.

The first part of the book dealt with the identification of hinging hyperplane based regression trees. The results of the developed algorithm prove that the implementation of a priori constraints enables the fuzzy c-regression clustering technique to identify hinging hyperplane models. The application of this technique recursively on the partitioned input space results in a regression tree capable of modeling and even of implementing model predictive control of technological data coming from real-life applications. The next section deals with the validation, visualization and structural reduction of neural networks. We describe in detail how the hidden layer of the neural network can be transformed to an additive fuzzy rule base.

T. Kenesei and J. Abonyi, *Interpretability of Computational Intelligence-Based Regression Models*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-21942-4_5

This section is followed by a description of connections between fuzzy regression and support vector regression, and the introduction of a three-step reduction algorithm to get interpretable fuzzy regression models on the basis of support vector regression.

We showed that hinging hyperplane models are excellent tools for the identification of models based on technological data. We tailored a new model structure on the hierarchical representation of hinging hyperplane models and delivered a new identification algorithm based on fuzzy clustering.

- To overcome the problems of the original hinge hyperplane identification algorithm delivered by Breimann, we adapted a fuzzy c-regression clustering algorithm for hinge identification by incorporating a priori constraints.
- As a further enhancement of this algorithm, we developed the hierarchical hinge hyperplane based on the regression tree identification technique. We showed the performance of the developed tool on multiple examples from well-known repositories.
- We proved that the identified transparent and interpretable models, with the help of the developed algorithm, are suitable for solving the process control duties of technological systems. To illustrate this feature we presented model predictive control of a simulated cartridge water heater.

To reinforce support vector regression methods, we worked out a three-step reduction technique in order to reduce and transform the support vector model into an interpretable fuzzy rule base. Further reduction of this rule base implies interpretable and robust regression models.

- We examined structural equivalency between support vector and fuzzy regression. We worked out a technique with the help of Gaussian kernels to transform the identified support vector model into a fuzzy rule base.
- Based on the identified support vector regression model, the transformed fuzzy rule base generates a large number of rules, making model interpretation and validation difficult. We tailored a three-step reduction algorithm to overcome this problem. We used the reduced set method to select the important set of support vectors and we utilized further similarity-based reduction of the generated rule base. The resultant fuzzy rule base is linear in the consequent part; therefore, we applied the orthogonal least squares algorithm for further reduction.

The interpretability of neural network models can be achieved by transforming the hidden layer of the neural network into a fuzzy rule base and by using a special, self-developed visualization technique of this rule base. Based on the self-developed transformation and visualization technique, we reduced the generated model with orthogonal least squares and similarity-based reduction techniques in order to support proper model structure design.

- We examined how validation and interpretability of black-box neural network models can be improved by transforming the hidden layer of the neural network models with a special operator to a fuzzy rule base.

- We compared calculated membership functions based on a similarity measure, enabling the analysis of the neural network model and pointing out possible further model reductions. This model structure is also linear in parameters from the output layer point, so we used the mentioned orthogonal least squares technique for further model reduction.
- Visualization of the neurons taking place on the hidden layer of a neural network can be achieved by distance measure and multidimensional scaling. This technique is also a new tool to examine and validate the structure of the neural network. The performance of these techniques is shown through a technological pH process.

The motivation to write our book was to integrate data, prior knowledge and extracted information into a single framework that helps model-building procedures with interpretability, visualization and reduction. The utilization of the developed algorithms was shown by sectionwise examples taken from the area of engineering. Benchmarks and experimental data were used to perform a comprehensive test of novel methods.

Due to its computational efficiency and easy interpretation, the hierarchical representation of the hinging hyperplane model proved to be a promising tool to develop local linear controllers. Interpretable fuzzy regression models initialized by robust support vector regression could help when, besides quantitative relationships, qualitative analysis is needed as well. The interpretable property of fuzzy models is a great vehicle for variable quality characterization. Structural validation and visualization of neural network models can support modellers address the challenge when only black-box model identification is possible. Our self-developed technique gives excellent feedback to determine model structure and evaluate task complexity. In the chemical industry, these problems occur when trying to find connections between complex reaction kinetic relationships and key technology and product quality variables.

Future developments from this book can branch in various directions in the field of interactive learning, where modeller experience combined with the learning capability of different identification techniques can lead to further successes.

Appendix A
n-Fold Cross Validation

This technique is intended to avoid the possible bias introduced by relying on any one particular division into test and train components by partitioning the original data in several different ways and computing the average of the performances over the different partitions. When the available data is divided into n parts, this approach is called *n-fold cross-validation*. Because of the n identification and verification steps, this method is computationally expensive. An extreme variant of this is to split the N training data points into a training set of size $N-1$ and a test set of size 1 and average the squared error on the left-out pattern over the N possible ways of obtaining such partition.

The leave-one-out validation (LOO) can be calculated analytically for linear-in-parameter models [95]. The fuzzy model is linear in its consequent parameters. Hence, the LOO criteria and its derivatives can be easily used for these models:

$$\hat{\sigma}^2_{LOO} = \frac{\mathbf{y}^T \mathbf{P} \left(\text{diag}(\mathbf{P})\right)^{-2} \mathbf{P}\mathbf{y}}{N}, \tag{A.1}$$

where in the case of global identification, $\mathbf{P}$ denotes the projection matrix

$$\begin{aligned} \mathbf{y} - \widehat{\mathbf{y}} &= \mathbf{y} - \mathbf{Q}\theta && \text{(A.2)} \\ &= \mathbf{y} - \mathbf{Q}\left(\mathbf{Q}^T\mathbf{Q}\right)^{-1}\mathbf{Q}^T\mathbf{y} && \text{(A.3)} \\ &= \left(\mathbf{I}_N - \mathbf{Q}\left(\mathbf{Q}^T\mathbf{Q}\right)^{-1}\mathbf{Q}^T\right)\mathbf{y} && \text{(A.4)} \\ &= \mathbf{P}\mathbf{y}, && \text{(A.5)} \end{aligned}$$

where the $\mathbf{Q}$ matrix contains the N regressors, $\mathbf{y}$ denotes the estimated outputs of the model, and $\mathbf{P}\widehat{\mathbf{y}}$ denotes the vector of the modeling error. The matrix $diag(\mathbf{P})$ is the same size and has the same diagonal as $\mathbf{P}$, but it is zero off-diagonal, and $\mathbf{I}_N$ represents an identity matrix.

T. Kenesei and J. Abonyi, *Interpretability of Computational Intelligence-Based Regression Models*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-21942-4

Appendix B
Orthogonal Least Squares

An often applied solution is to prune the identified model trained with the classical cost function. In the following, model reduction techniques of this type will be considered. In general, it can be stated that linear model reduction methods are preferred to nonlinear ones because they are exhaustively studied and effectively applied for several types of problems. For that purpose the model should be linear in parameters. A possible method family is that of orthogonal techniques. These methods can roughly be divided into two groups: the rank-revealing ones, such as the SVD-QR algorithm, and those that evaluate the individual contribution of the rule or local models, such as the orthogonal least squares approach (OLS). This latter technique requires more computations, but for system identification purposes it is preferable as it gives a better approximation result. In this book OLS is applied for rule ranking and model reduction purposes. OLS works as follows (for a throughout discussion see [1]). Consider a general linear-in-parameters model;

$$\mathbf{y} = \mathbf{Z}\theta + \mathbf{e}, \tag{B.1}$$

where $\mathbf{y} = [y_1, \ldots, y_N]^T$ is the measured output, $\mathbf{Z} = [\mathbf{z_1}, \ldots, \mathbf{z_n}]^T$ is the regressor matrix ($\mathbf{z_i} = [z_{i1}, \ldots, z_{iN}]^T$, $i = 1, \ldots, h$ are the regressors), $\theta = [\theta_1, \ldots, \theta_h]$ is the parameter vector and $\mathbf{e} = [e_1, \ldots, e_N]^T$ is the prediction error. OLS transforms the columns of the regressor matrix $\mathbf{Z}$ into a set of orthogonal basis vectors in order to inspect the individual contribution of each regressor. If they were not orthogonal, they could not have been inspected individually. An orthogonalization method should be used to perform the orthogonal decomposition $\mathbf{Z} = \mathbf{VR}$ (often the simple Gram-Schmidt method is used), where $\mathbf{V}$ is an orthogonal matrix such that $\mathbf{V}^T\mathbf{V} = \mathbf{I}$ and $\mathbf{R}$. Substituting $\mathbf{Z} = \mathbf{VR}$ into Eq. B.1, we get $\mathbf{y} = \mathbf{VR}\theta + \mathbf{e} = \mathbf{Vg} + \mathbf{e}$, where $\mathbf{g} = \mathbf{R}\theta$. Since the columns $\mathbf{v_i}$ of $\mathbf{V}$ are orthogonal, the sum of squares of y_k can be written as

T. Kenesei and J. Abonyi, *Interpretability of Computational Intelligence-Based Regression Models*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-21942-4

$$\mathbf{y}^T\mathbf{y} = \sum_{i=1}^{h} g_i^2 \mathbf{v}_i^T \mathbf{v}_i + \mathbf{e}^T\mathbf{e}. \tag{B.2}$$

The part of the output variance $\mathbf{y}^T\mathbf{y}/N$ explained by regressors is $\sum_{i=1}^{h} g_i^2 \mathbf{v}_i^T \mathbf{v}_i/N$ and an error reduction ratio due to an individual regressor i can be defined as

$$err_i = \frac{g_i^2 \mathbf{v}_i^T \mathbf{v}_i}{\mathbf{y}^T\mathbf{y}},\ i = 1, \ldots, h. \tag{B.3}$$

This ratio offers a simple means for ordering the regressors. As [1] shows, "there are only two restrictions to the application of this subset selection technique. First, the model has to be linear in parameters. Second, the set of regressors from which the significant ones will be chosen must be precomputed." This latter restriction is an important one because it means that all regressors are fixed during this procedure. This requirement is not met by normalized RBF networks and Takagi-Sugeno fuzzy models; therefore the original version of OLS cannot be applied. It is because the normalization denominator changes with the number of selected rules; thus the fuzzy basis functions (here: regressors) change. To overcome this problem, the value of the denominator can be fixed, but in this case interpretability issues are discarded completely. However, OLS can be very useful for various purposes; modified versions of OLS can also be applied to determine the centers of radial basis functions, or to generate Takagi-Sugeno-Kang fuzzy models.

Appendix C
Model of the pH Process

The modeling and control of pH (the concentration of hydrogen ions) in a continuously stirred tank reactor (CSTR) is a well-known control problem that presents difficulties due to the nonlinearity of the process dynamics. The CSTR is shown schematically in Fig. C.1.

A dynamic model of the pH in a tank can be obtained by considering the material balances on $[Na^+]$ and the total acetate $[HAC + AC^-]$ and assuming that the acid-base equilibrium and electroneutrality relationships hold [96].

Total acetate balance:

$$F_{HAC}[HAC]_{in} - (F_{HAC} + F_{NaOH})[HAC + AC^-] = V\frac{d[HAC + AC^-]}{dt}.$$

Sodium ion balance:

$$F_{NaOH}[NaOH]_{in} - (F_{HAC} + F_{NaOH})[Na^+] = V\frac{d[Na^+]}{dt}.$$

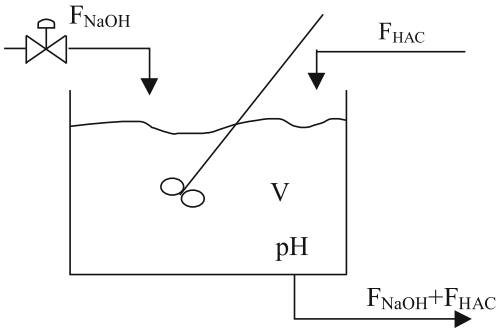

Fig. C.1 Scheme of the pH setup

T. Kenesei and J. Abonyi, *Interpretability of Computational Intelligence-Based Regression Models*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-21942-4

Table C.1 Parameters used in the simulations

Parameter	Description	Nominal value
V	Volume of the tank	1000 [l]
F_{HAC}	Flow rate of acetic acid	81 [l/min]
F_{NaOH}	Flow rate of NaOH	515 [l/min]
$[NaOH]_{in}$	Inlet conc. of NaOH	0.05 [mol/l]
$[HAC]_{in}$	Inlet conc. of acetic acid	0.32 [mol/l]
$[Na^+]$	Initial conc. of sodium in the CSTR	0.0432 [mol/l]
$[HAC + AC^-]$	Initial conc. of acetate in the CSTR	0.0432 [mol/l]
K_a	Acid equilibrium constant	1.75310^{-5}
K_w	Water equilibrium constant	10^{-14}

HAC equilibrium:

$$\frac{[AC^-][H^+]}{[HAC]} = K_a.$$

Water equilibrium:

$$[H^+][OH^-] = K_w.$$

Electroneutrality:

$$[Na^+] + [H^+] = [OH^-] + [AC^-].$$

The pH can be calculated from the previous equations as

$$[H^+]^3 + [H^+]^2(K_a + [Na^+]) + [H^+]([Na^+]K_a - [HAC + AC^-]K_a - K_w) - K_w K_a = 0,$$

$$pH = -\log[H^+].$$

The parameters used in our simulations are taken from [97] and are given in Table C.1.

Appendix D
Model of Electrical Water Heater

The schematic diagram of the water heater is shown in Fig. D.1.

The water comes from the water pipeline into the heater through a control valve and a pair of metal pipes containing a cartridge heater. The control task is to control the T_{out} outlet temperature by adjusting the u heating signal of the cartridge heater.

The temperature measurement is realized using Pt100 thermometers. The system has four analog inputs (inlet temperature T_{in}, outlet temperature T_{out}, valve position, and the flow rate F), and two digital (open and close in the valve, or CVO and

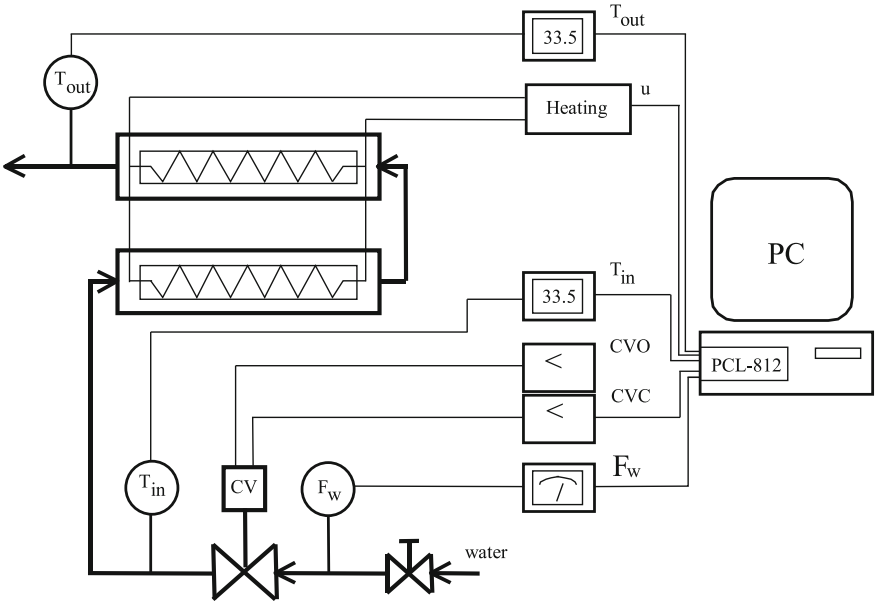

Fig. D.1 The scheme of the physical system

T. Kenesei and J. Abonyi, *Interpretability of Computational Intelligence-Based Regression Models*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-21942-4

CVC) and one analog output (heating control signal, u). The heaters are linked in parallel and have a performance of 1 kW. The process is connected to a PC computer through ADVANTECH LabCard PCLD-780 and PCL-812 data acquisition boards. The GENIE 3.02 data acquisition and control software was used to filter and convert the input signals (0–5 V). The control algorithm runs in MATLAB 4.2. The sampling time of the control system is seconds [98].

For the purpose of physical modeling, the system was decomposed into four interacting elements: the cartridge heater (subscript h), the streaming water (subscript w), the pipe wall (subscript p) and the environment (subscript e). The following three heat balances in the form of partial differential equations can be established:

$$
\begin{aligned}
V_h \rho_h C_{p_h} \frac{\partial T_h}{\partial t}(t,z) &= Q(u) - \alpha_1 A_1 (T_h - T_w), \\
V_w \rho_w C_{p_w} \frac{\partial T_w}{\partial t}(t,z) + (F \rho C_p)_w \frac{\partial T_w}{\partial z}(t,z) &= \alpha_1 A_1 (T_h - T_w) - \alpha_2 A_2 (T_w - T_p), \\
V_p \rho_p C_{p_p} \frac{\partial T_p}{\partial t}(t,z) &= \alpha_2 A_2 (T_w - T_p) - \alpha_e A_e (T_p - T_e),
\end{aligned}
$$

with $z \in [0, L]$ where L denotes the length of the pipe. The description and the nominal values of the parameters are given in Table D.1.

Table D.1 Parameters used in the simulation model of the heating system

Parameter	Description	Nominal value
L	Length of the pipe	2×48010^{-3} m
ρ_h	Density of the cartridge	3650 kg/m^3
C_{ph}	Heat capacity of the cartridge	1047 J/kg K
A_h	Surface of the cartridge	2.41×10^{-2} m^2
V_h	Volume of the cartridge	4.82×10^{-5} m^3
α_1	$h - w$ heat transfer coefficient	316.3 W m^{-2} K^{-1}
ρ_w	Density of the water	1000 kg/m^3
C_{pw}	Heat capacity of the water	4186 J/kg K
T_{in}	Inlet water temperature	11.8 °C
V_w	Volume of the water	1.16×10^{-4} m^3
α_2	$w - p$ heat transfer coefficient	1196.1 W m^{-2} K^{-1}
ρ_p	Density of the wall	7850 kg/m^3
C_{pp}	Heat capacity of the wall	502 J/kg K
t_e	Temperature of the environment	21.6 °C
A_p	Inner surface of the wall	4.46×10^{-2} m^2
V_p	Volume of the wall	7.37×10^{-5} m^3
A_e	Outer surface of the wall	5.36×10^{-2} m^2
α_e	$p - e$ heat transfer coefficient	1015.9 W m^{-2} K^{-1}

The performance of the cartridge heater is given by

$$Q(u) = Q_M \left[u - \frac{\sin(2\pi u)}{2\pi} \right], \tag{D.1}$$

where Q_M is the maximal power, and u is the heating signal (voltage). The partial differential equations are approximated by eight compartments of equal volume. As Eq. D.1 shows, the heating performance is a static nonlinear function of the heating signal (control input).

References

1. Nelles, O. (2001). *Nonlinear system identification*. New York: Springer.
2. Mamdani, E. H., Teraqno, T., Asai, K., & Sugeno, M. (1992). Fuzzy-systems theory and its applications. *Nature*, *359*, 788–788.
3. Mendel, J. M. (1995). Fuzzy logic systems for engineering: A tutorial. *Proceedings of the IEEE*, *83*, 345–377.
4. Bhat, N. V., & McAvoy, T. J. (1990). Use of neural nets for dynamic modelling and control of chemical process systems. *Computers and Chemical Engineering*, *4*(5), 573–585.
5. Bhat, N. V., Minderman, P. A., McAvoy, T. J., & Wang, N. S. (1990, April 24–30). Modelling chemical process systems via neural computation. *IEEE Control Systems Magazine*.
6. Hernandez, E., & Arkun, Y. (1993). Control of nonlinear systems using polynomial ARMA models. *AICHE Journal*, *39*(3), 446–460.
7. Ydstie, B. E. (1990). Forecasting and control using adaptive connectionist networks. *Computers and Chemical Engineering Journals*, *15*(4/5), 583–599.
8. Sjöberg, J., Zhang, Q., Ljung, L., Benveniste, A., Deylon, B., Glorennec, P.-Y., et al. (1995). Nonlinear black-box modeling in system identification: A unified overview. *Automatica*, *31*, 1691–1724.
9. Abonyi, J., & Feil, B. (2005). Computational intelligence in data mining. *Informatica*, *29*, 3–12.
10. Piatetsky-Shapiro, G. (1991). *Knowledge discovery in databases*. Cambridge, MA: AAAI Press.
11. Zadeh, L. A. (1973). Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions and Systems, Man, and Cybernetics*, *3*, 28–44.
12. Sugeno, M., & Kang, G. T. (1988). Structure identification of fuzzy model. *Fuzzy Sets and Systems*, *28*, 15–33.
13. Sugeno, M., & Tanaka, K. (1991). Successive identification of a fuzzy model and its application to prediction of a complex system. *Fuzzy Sets and Systems*, *42*, 315–334.
14. Sugeno, M., & Yasukawa, T. (1993). A fuzzy-logic-based approach to qualitative modeling. *IEEE Transactions on Fuzzy Systems*, *1*(1), 7–31.
15. Jang, J. -S. R. (1996). Input selection for ANFIS learning. In *Proceedings of the IEEE International Conference on Fuzzy Systems* (Vol. 2, pp. 1493–1499). New York, USA.
16. Zadeh, L. A. (1994). Fuzzy logic, neural networks, and soft computing. *Communications of the ACM*, *36*(3), 77–84.
17. Kecman, V. (2001). *Learning and Soft Computing*.

T. Kenesei and J. Abonyi, *Interpretability of Computational Intelligence-Based Regression Models*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-21942-4

18. Niu, S., & Pucar, P. (1995). Hinging hyperplanes for non-linear identification. http://www.control.isy.liu.se.
19. Ernst, S. (1998). Hinging hyperplane trees for approximation and identification. In *Proceedings of the 37th IEEE Conference on Decision and Control* (Vol. 2, pp. 1266–1271).
20. Breiman, L. (1993). Hinging hyperplanes for regression, classification and function approximation. *IEEE Transactions on Information Theory*, *39*, 311–325.
21. Nunez, H., Angulo, C., Catala, A. (2002). Rule extraction from support vector machines. In *European Symposium on Artificial Neural Networks Proceedings* (pp. 107–112).
22. Chen, Y. X., & Wang, J. Z. (2003). Support vector learning for fuzzy rule-based classification systems. *IEEE Transactions on Fuzzy Systems*, *11*, 716–728.
23. Chen, Y. X., & Wang, J. Z. (2003). Kernel machines and additive fuzzy systems: Classification and function approximation. In *Proceedings of IEEE International Conference on Fuzzy Systems* (pp. 789–795).
24. Cortes, C., & Vapnik, V. (1995). *Support-vector networks*. USA: AT&T Research Labs.
25. Jacek, Q., & Leski, M. (2006). On support vector regression machines with linguistic interpretation of the kernel matrix. *Fuzzy Sets and Systems*, *157*, 1092–1113.
26. Cun, Y. L., Denke, J., & Solla, S. (1990). Optimal brain damage. *Advances in Neural Information Processing Systems*, *2*, 598–605.
27. Duch, W. (2003). Coloring black boxes: Visualization of neural network decisions. *The International Joint Conference on Neural Networks Portland*, *1*, 1735–1740.
28. Duch, W. (2004). Visualization of hidden node activity in neural networks: I. visualization methods. *Lecture Notes in Artificial Intelligence*, *3070*, 38–43.
29. Duch, W. (2004). Visualization of hidden node activity in neural networks: II. application to RBF networks. *Lecture Notes in Artificial Intelligence*, *3070*, 44–49.
30. Henrique, H. M., Lima, E. L., & Seborg, D. E. (2000). Model structure determination in neural network models. *Chemical Engineering Science*, *55*, 5457–5469.
31. Benitez, J. M., Castro, J. L., & Requena, I. (1995). Are artifical neural networks black boxes? *IEEE Transactions on Neural Networks*, *8*(5), 1156–1164.
32. Abonyi, J. (2003). *Fuzzy model identification for control*. Boston: Birkhäuser.
33. Ramirez, D. R., Camacho, E. F., & Arahal, M. R. (2004). Implementation of Min-Max MPC using hinging hyperplanes to a heat exchanger. *Control Engineering Practice*, *12*, 1197–1205.
34. Pucar, P., & Sjoberg, J. (1995). Parameterization and conditioning of hinging hyperplane models. In *Technical Report LiTH-ISY-R-1809* (Vol. 40, pp. 37–50), Department of Electrical Engineering, Linkoping University.
35. Wen, C., Wang, S., Jin, X., & Ma, X. (2007). Identification of dynamic systems using piecewise-affine basis function models. *Automatica*, *43*, 1824–1831.
36. Roll, J., Bemporad, A., & Ljung, L. (2004). Identification of piecewise affine systems via mixed-integer programming. *Automatica*, *40*, 37–50.
37. Hathaway, R., & Bezdek, J. (1993). Switching regression models and fuzzy clustering. *IEEE Transactions on fuzzy systems*, *3*, 195–204.
38. Abonyi, J., & Feil, B. (2007). *Cluster analysis for data mining and system identification*. Basel: Birkhäuser.
39. Sala, A., Guerrab, T. M., & Babuska, R. (2005). Perspectives of fuzzy systems and control. *Fuzzy Sets and Systems*, *156*, 432–444.
40. Hellendoorn, H., & Precup, R. (2011). A survey on industrial applications of fuzzy control. *Computers in Industry*, *63*, 213–226.
41. Feng, G. (2011). A survey on analysis and design of model-based fuzzy control systems. *IEEE Transactions on Fuzzy Systems*, *14*(5), 676–697.
42. Babuška, R. (1998). *Fuzzy Modeling for Control*. Boston: Kluwer Academic Publishers.
43. Abonyi, J., Babuska, R., Setnes, M., Verbruggen, H. B., & Szeifert, F. (1999). Constrained parameter estimation in fuzzy modeling. In *Proceedings FUZZ-IEEE'99* (pp. 951–956).
44. Abonyi, J., Babuska, R., Szeifert, F., & Nagy, L. (2000). Identification and control of nonlinear systems using fuzzy Hammerstein models. *Industrial and Engineering Chemistry Research*, *39*(11), 4302–4314.

45. Kubat, M. (1998). Decision trees can initialize radial-basis-function networks. *IEEE Transactions on Neural Networks*, *9*, 813–821.
46. Wen, C., & Ma, X. (2008). A max-piecewise-linear neural network for function approximation. *Neurocomputing*, *71*, 843–852.
47. Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.
48. Pucar, P., & Sjoberg, J. (1998). On the hinge finding algorithm for hinging hyperplanes. *IEEE Transaction on Information Theory*, *44*, 1310–13109.
49. Bemporad, A., Garulli, A., Paoletti, S., & Vicino, A. (2005). A bounded-error approach to piecewise affine system identification. *IEEE Transactions on Automatic Control*, *50*, 1567–1280.
50. Friedman, J. H. (1991). Multivariate adaptive regression splines. *The Annals of Statistics*, *19*, 1–141.
51. Cai, Q., Hao, Z., & Yang, X. (2012). Gaussian kernel-based fuzzy inference systems for high dimensional regression. *Neurocomputing*, *77*, 197–204.
52. Weierstrass, K. (1872). Über continuirliche Functionen eines reellen Arguments die für keinen Werth des letzeren einen bestimmten Differentialquotienten besitzen. Königliche Akademie der Wissenschaften.
53. Weierstrass, K. (1885). *Über die analytische Darstellberkeit sogenannter willkürlicher Functionen einer reellen veränderlichen*. Berlin: Sitzungsberichte der Königlich Preussischen Akademie der Wissenschaften zu.
54. Hilbert, D. (1900). Mathematische Probleme. *2. International Congress of Mathematicians*, Paris, France.
55. Arnold, V. I. (1957). On functions of three variables. *Doklady Akademii Nauk USSR*, *114*, 679–681.
56. Kolmogorov, A. N. (1957). On the representation of continuous functions of many variables by superpositions of continuous functions of one variable and addition. *Doklady Akademii Nauk USSR*, *114*, 953–956.
57. Sprecher, D. A. (1965). On the structure of continuous functions of several variables. *Transactions of the American Mathematical Society*, *115*, 340–355.
58. Lorentz, G. G. (1966). *Approximation of functions*.
59. De Figueiredo, J. P. (1980). Implications and applications of Kolmogorov's superposition theorem. *IEEE Transactions on Automated Control*, *25*(6), 1127–1231.
60. Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*, 359–366.
61. Blum, E. K., & Li, L. K. (1991). Approximation theory and feedforward networks. *Neural Networks*, *4*(4), 511–515.
62. Kurková, V. (1992). Kolmogorov's theorem and multilayer neural networks. *Neural Networks*, *5*, 501–506.
63. Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*, 338–353.
64. Wang, L. X. (1992). Fuzzy systems are universal approximators. In *IEEE International Conference on Fuzzy Systems* (pp. 1163–1169). San Diego, USA.
65. Kosko, B. (1994). Fuzzy systems are universal approximators. *IEEE Transactions on Computers*, *43*(11), 1329–1333.
66. Castro, J. L. (1995). Fuzzy logic controllers are universal approximators. *IEEE Transactions on SMC*, *25*, 629–635.
67. Moser, B. (1999). Sugeno controllers with a bounded number of rules are nowhere dense. *Fuzzy Sets and Systems*, *104*(2), 269–277.
68. Tikk, D. (1999). On nowhere denseness of certain fuzzy controllers containing prerestricted number of rules. *Tatra Mountains Mathematical Publications*, *16*, 369–377.
69. Klemet, E. P., Kóczy, L. T., & Moser, B. (1999). Are fuzzy systems universal approximators? *International Journal of General Systems*, *28*(2), 259–282.
70. Stone, M. H. (1937). Applications of the theory of boolean rings to general topology. *Transactions of the American Mathematical Society*, *41*(3), 375–481.

71. Stone, M. H. (1948). The generalized Weierstrass approximation theorem. *Mathematics Magazine*, *21*(4), 167–184.
72. Cotter, N. E. (1990). The Stone-Weierstrass theorem and its application to neural networks. *IEEE Transaction on Neural Networks*, *1*(4), 290–295.
73. Wang, L. X., & Mendel, J. M. (1992). Fuzzy basis functions, universal approximators and orthogonal least squares learning. *IEEE Transactions on Neural Networks*, *3*, 807–814.
74. Setiono, R., Leow, W. K., & Thong, J. Y. L. (2000). Opening the neural network black box: An algorithm for extracting rules from function approximating artificial neural networks. In *Proceedings of the 21st International Conference on Information Systems* (pp. 176–186). Queensland.
75. de Castro, L. N., Iyoda, E. M., Zuben, F. J. V., & Gudwin, R. (1998). Feedforward neural network initialization: An evolutionary approach. In *Proceedings of the 5th Brazilian Symposium on Neural Networks* (pp. 43–49).
76. Chumieja, M., & Markowska-Kaczmar, U. (2003). Opening neural network black box by evolutionary approach. In *Design and Application of Hybrid Intelligent Systems* (pp. 147–156).
77. Bath, N. V., & McAvoy, T. J. (1992). Determining model structure for neural models by network stripping. *Computers and Chemical Engineering*, *16*(4), 271–281.
78. Wolff, G., Hassibi, B., & Stork, D. (1992). Optimal brain surgeon and general network pruning. *Technical report*.
79. Chiang, J. H., & Hao, P. Y. (2004). Support vector learning mechanism for fuzzy rule-based modeling: A new approach. *IEEE Transactions on Fuzzy Systems*, *12*, 1–12.
80. Yang, X., Cai, Q., & Haom, Z. (2011). Gaussian kernel-based fuzzy inference system for high dimensional regression. *Neurocomputing*.
81. Castro, L., Flores-Hidalgo, L. D., Mantas, C. J., & Puche, J. M. (2007). Extraction of fuzzy rules from support vector machines. *Fuzzy Sets and Systems*, *158*(18), 2057–2077.
82. Hong, D. H., & Hwang, C. (2003). Support vector fuzzy regression machines. *V*, *138*(2), 271–281.
83. Gunn, S. R. (1998). Support vector machines for classification and regression. *Technical report*.
84. Juang, C.-F., & Hsieh, C.-D. (2009). TS-fuzzy system-based support vector regression. *Fuzzy Sets and Systems*, *160*(17), 2486–2504.
85. Celikyilmaz, A., & Türksen, I. B. (2007). Fuzzy functions with support vector machines. *Information Sciences*, *177*(23), 5163–5177.
86. Wang, H.-F., & Tsaur, R.-C. (2000). Insight of a fuzzy regression model. *Fuzzy Sets and Systems*, *112*(3), 355–369.
87. Üstün, B., Buydens, L. M. C., & Melssen, W. J. (2007). Visualisation and interpretation of support vector regression models. *Analytica Chimica Acta*, *595*(1–2), 299–309.
88. Schölkopf, B., Mika, S., Burges, C. J. C., Knirsch, P., Üller, K.-R. M., Ratsch, G., & Smola., A. (1999). Input space vs. feature space in kernel-based methods. *IEEE Transactions on Neural Networks*, *10*(5).
89. Setnes, M., Babuska, R., Kaymak, U., & van Nauta Lemke, H. R. (1998). Similarity measures in fuzzy rule base simplification. *IEEE Transactions on SMC-B*, *28*, 376–386.
90. Yen, J., & Wang, L. (1999). Simplifying fuzzy rule-based models using orthogonal transformation methods. *IEEE Transactions on SMC-B*, *29*, 13–24.
91. Setnes, M., & Hellendoorn, H. (2000). Orthogonal transforms for ordering and reduction of fuzzy rules. In *FUZZ-IEEEE* (p. 700.705). San Antonio, Texas.
92. Smola, A. M., & Schölkopf, B. (2003). A tutorial on support vector regression. *NeuroCOLT Tehnical Report*.
93. Jin, Y. (2000). Fuzzy modeling of high-dimensional systems. *IEEE Transactions on Fuzzy Systems*, *8*, 212–221.
94. Setnes, M., & Babuška, R. (2001). Rule base reduction: Some comments on the use of orthogonal transforms.
95. Orr, M. J. L. (1996, April). Introduction to radial basis function networks. In *Research Report*. Centre of Cognitive Science, University of Edinburgh.

96. McAvoy, T. J., Hsu, E., & Lowenthal, S. (1972). Dynamics of pH in controlled stirred tank reactor. *Industrial Engineering Chemical Process Design and Development*, *11*(1), 68–70.
97. Bhat, N. V., & McAvoy, T. J. (1992). Determining model structure for neural models by network stripping. *Computers and Chemical Engineering*, *16*, 271–281.
98. Abonyi, J., Bodizs, A., Nagy, L., & Szeifert, F. (1999) Hybrid fuzzy convolution model based predictor corrector controller. In M. Mohamadian (Ed.), *Computational intelligence for modelling control and automation* (pp. 265–270). IOS Press, ISBN 9-051-99474-5.

Index

T. Kenesei and J. Abonyi, *Interpretability of Computational Intelligence-Based Regression Models*, SpringerBriefs in Computer Science,
DOI 10.1007/978-3-319-21942-4