TREE ENSEMBLES FOR DISCRETE-TIME MODELING OF NON-LINEAR DYNAMIC SYSTEMS

Darko Aleksovski

Doctoral Dissertation
Jožef Stefan International Postgraduate School
Ljubljana, Slovenia

Supervisor: Prof. Dr. Sašo Džeroski, Jožef Stefan Institute, Ljubljana, Slovenia
Co-Supervisor: Prof. Dr. Juš Kocijan, Jožef Stefan Institute, Ljubljana, Slovenia

Evaluation Board:
Prof. Dr. Đani Juričić, Chair, Jožef Stefan Institute, Ljubljana, Slovenia
Asst. Prof. Dr. Bernard Ženko, Member, Jožef Stefan Institute, Ljubljana, Slovenia
Prof. Dr. Ljupčo Todorovski, Member, Faculty of Administration, University of Ljubljana, Ljubljana, Slovenia

MEDNARODNA PODIPLOMSKA ŠOLA JOŽEFA STEFANA
JOŽEF STEFAN INTERNATIONAL POSTGRADUATE SCHOOL

Darko Aleksovski

TREE ENSEMBLES FOR DISCRETE-TIME MODELING OF NON-LINEAR DYNAMIC SYSTEMS

Doctoral Dissertation

ANSAMBLI DREVES ZA MODELIRANJE NELINEARNIH DINAMIČNIH SISTEMOV V DISKRETNEM ČASU

Doktorska disertacija

Supervisor: Prof. Dr. Sašo Džeroski

Co-Supervisor: Prof. Dr. Juš Kocijan

Ljubljana, Slovenia, September 2014

Acknowledgments

I would like to show my gratitude to my supervisor Prof. Dr. Sašo Džeroski, my co-supervisor Prof. Dr. Juš Kocijan, the Jožef Stefan International Postgraduate School, and the Jožef Stefan Institute, for providing me with an excellent environment for work on the thesis. I would also like to thank my thesis committee, Prof. Dr. Đani Juričič, Prof. Dr. Ljupčo Todorovski, and Asst. Prof. Dr. Bernard Ženko, for their helpful comments and suggestions.

Several institutions provided funding for my research work, the Slovene Human Resources Development and Scholarship Fund, the Department of Knowledge Technologies (Jožef Stefan Institute, Ljubljana), and the European Commission (through the projects PHAGOSYS: Systems biology of phagosome formation and maturation - modulation by intracellular pathogens; SUMO: Supermodeling by combining imperfect models; and MAESTRA: Learning from Massive, Incompletely annotated, and Structured Data).

I am especially grateful to my wife, my parents, and my sister, for their constant and unconditional support. I would also like to thank everyone who helped me write this dissertation successfully.

Abstract

In the thesis, we address the problem of discrete-time modeling of non-linear dynamic systems. Using the external dynamics approach, the problem is transformed to a nonlinear static regression function approximation problem. We consider multiple-model approaches, which build several simple models.

We build upon one crisp general-purpose approach, which builds crisp linear model trees, and one fuzzy linear model tree approach. The former is known to be fast, but has not been evaluated for modeling dynamic systems. While the latter is slower, its applications for modeling dynamic systems show that it is rather accurate. We evaluate and further develop the two approaches to improve the predictive performance of the first, and the running time of the second approach.

In particular, we consider the crisp model tree algorithm M5' and the fuzzy model tree algorithm Lolimot. The models they build differ: the first builds crisp model trees, while the second builds fuzzy model trees, also known by the name of Takagi-Sugeno models. This thesis also investigates the use of ensembles built on top of the two tree learning algorithms, as they would potentially be able to increase the predictive performance. The ensemble approach considered is bagging, aimed at reducing the variance of the model. We also consider multi-output trees, which were introduced earlier, but not yet evaluated and popularized, and ensembles thereof.

We empirically evaluate the approaches of building model trees and ensembles on a collection of measured and synthetic datasets derived from seven dynamic system case studies. In order to provide for a more realistic assessment of performance, the datasets used contain noise. The performance results obtained are analyzed using statistical tests. The results of the empirical evaluation show that ensembles improve over the performance of single model trees, and that the multi-output tree approach is worth considering, as it decreases the complexity of the overall multi-output models. They also show that the modifications of the Lolimot algorithm decrease its running time, while maintaining the same level of predictive performance.

The first contribution of this thesis are the modifications and analysis of two model tree learning algorithms, introduced in different domains, for the task of modeling dynamic systems. Also, this thesis provides an implementation and evaluation of ensemble approaches based on the two tree learning algorithms. The ensembles of model trees are evaluated on static regression tasks. Finally, both single trees and ensembles are evaluated on single-output and multi-output dynamic system case studies, which provides insights into their performance.

Povzetek

Disertacija obravnava problem modeliranja nelinearnih dinamičnih sistemov v diskretnem času. S pristopom zunanje dinamike smo problem pretvorili v regresijski problem aproksimacije nelinearne statične funkcije. V disertaciji smo upoštevali večmodelne metode modeliranja, kjer so modeli sestavljeni iz več enostavnih podmodelov.

V disertaciji smo nadgradili splošnonamensko metodo za modeliranje, ki gradi drevesa linearnih modelov, in metodo mehkih dreves linearnih modelov. Prva metoda je znana po svoji hitrosti učenja modela, vendar do sedaj še ni bila preizkušena za modeliranje dinamičnih sistemov. Medtem ko je druga metoda pri učenju počasnejša, se je do sedaj izkazala kot precej natančnejša za modeliranje dinamičnih sistemov. Omenjeni metodi smo ovrednotili in nadgradili tako, da smo izboljšali napovedno točnost modelov, dobljenih s prvo metodo, in čas potreben za učenje modelov z drugo metodo.

Obravnavani metodi sta bili metoda modelskih dreves M5' in metoda mehkih modelskih dreves Lolimot. Modeli, ki jih metodi gradijo, se razlikujejo: prva gradi drevesa linearnih modelov, medtem ko druga gradi mehka drevesa linearnih modelov, ki so znani tudi kot mehki model Takagi-Sugeno. Disertacija obravnava tudi drevesa za modeliranje sistemov z več izhodi. Taki modeli dreves so bili sicer že znani, vendar še ne ustrezno ovrednoteni ter uporabljeni v ansamblih, kar smo storili v disertaciji.

Metode gradnje dreves linearnih modelov in njihovih ansamblov smo ovrednotili empirično na množici merjenih in simuliranih podatkov sedmih primerov dinamičnih sistemov. Uporabljene množice podatkov vsebujejo šum in zaradi tega omogočajo bolj realistično vrednotenje dobljenih modelov. Rezultate vrednotenja smo analizirali s statističnimi testi. Rezultati empiričnega vrednotenja so pokazali, da so ansambli modelskih dreves bolj uspešni pri napovedovanju kot posamezna drevesa. Poleg tega je bilo razvidno tudi, da je smiselno uporabiti metodo modelskih dreves za modeliranje dinamičnih sistemov z več izhodi, saj se na ta način zmanjša kompleksnost modela v primerjavi z več modeli z enim izhodom. Rezultati vrednotenja so pokazali, da so predlagane izboljšave metode Lolimot zmanjšale čas učenja modela pri enaki napovedni točnosti.

Prvi prispevek disertacije so izboljšave in vrednotenja omenjenih dveh algoritmov za učenje regresijskih dreves za modeliranje dinamičnih sistemov, ki izhajata iz različnih strokovnih področij. Predstavili smo metode za gradnjo ansamblov obeh vrst dreves in njihovo vrednotenje. Disertacija vsebuje tudi vrednotenje ansamblov modelskih dreves za modeliranje statičnih regresijskih problemov. Zadnji prispevek je empirično ovrednotenje posameznih dreves in ansamblov na primerih z enim in več izhodi, ki daje vpogled v učinkovitost obravnavanih metod za modeliranje.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ODE	...	Ordinary differential equation
PDE	...	Partial differential equation
MT	...	Model tree
ANN	...	Artificial neural network
MLP	...	Multi-layer perceptron
RBF	...	Radial-basis function
SVR	...	Support vector regression
TS	...	Takagi-Sugeno, or Takagi-Sugeno-Kang fuzzy model
MSF	...	Membership function, fuzzy membership function
LM	...	Local model
TDIDT	...	Top-down induction of decision trees
RRMSE	...	Root-relative mean squared error
SDR	...	Standard deviation reduction
SO	...	Single-output
MO	...	Multiple-output, or multi-output
AIC	...	Akaike information criterion
OSA	...	One-step ahead prediction error
SIM	...	Simulation error, or output error
RT	...	Regression tree
MTE	...	Model tree ensemble
BMT	...	Bagging of model trees
FMT	...	Forest of model trees
BRT	...	Bagging of regression trees
FRT	...	Forest of regression trees
CSTR	...	Continuous-stirred tank reactor
GLS	...	Gas-liquid separator
MIMO	...	Multiple-input multiple-output system
MISO	...	Multiple-input single-output system

Symbols

D	...	The data points available for training
$n = \|D\|$	...	The number of data points in D
r	...	The number of target variables
m	...	The number of local models (LMs), i.e., terminal nodes of a model tree
p	...	The number of regressors (features used to build LMs)
t	...	The size of the the ensemble, i.e., number of trees in the ensemble
D_j	...	The j-th bootstrap sample of the set D
T_j	...	The j-th tree in the ensemble
μ	...	Fuzzy membership function
Φ	...	Fuzzy validity function

Chapter 1

Introduction

Dynamic systems, the state of which changes over time, are ubiquitous in both science and engineering. Experts build models of a dynamic system to analyze it, simulate and predict its behavior under various conditions. The system behavior is represented by time series of state variables values. In continuous time, the models of dynamic systems typically take the form of ordinary differential equations (ODEs) or partial differential equations (PDEs), where the rate of change of the state variable is expressed as a function of its current state, as well as the values of input (exogenous) variables[1].

In discrete time, the models of dynamic systems typically take the form of difference (recurrence) equations, which describe the next state of the system, using the current state and input variables. Using the external dynamics approach (Nelles, 2001), the discrete-time modeling of dynamic systems can be reformulated and solved as a regression problem.

In this thesis we consider multiple-model approaches for solving the regression problem mentioned. These approaches build several simpler models, each valid in its own region. In particular, we are interested in tree-based approaches, which are able to define the boundaries of the regions in a more efficient manner, as compared to other types of partitioning (e.g., grid partitioning). We limit our research only to multiple-model approaches based on trees that use linear models named model trees.

The model trees considered and used here are crisp (linear) model trees and fuzzy (linear) model trees. On the one hand, the crisp model trees utilize crisp binary splits, i.e., a data point is sorted down either the left or the right subtree of a split node. This implies that the final prediction of the model is calculated by using one terminal node only, i.e., one local linear model. On the other hand, the fuzzy model trees use fuzzy binary splits, which associate each data point with both subtrees of a split node with different weights. The final prediction of a fuzzy model tree is obtained by consulting all local models, each of them having different importance (w_i) for the final prediction.

Ensembles are frequently used to increase the predictive performance of the crisp (and fuzzy) tree-based approaches. The ensembles consist of several base models. The ensembles we consider, i.e., bagging and forests, perform bootstrap random resampling of training data and build a model on each sample: By combining their predictions, ensembles reduce the variance of base models.

[1] According to the continuous-time state equations. Please note that other fields may use different terminology. For example a system (endogenous) variable denotes a factor in a causal system whose value is determined by the states of other variables in the system.

1.1 Aims and Goals

The main goal of our research is to investigate the problem of discrete-time modeling of nonlinear dynamic systems and propose novel solutions based on linear model trees and ensembles thereof. The general-purpose crisp linear model trees, introduced in the domain of machine learning, have not been considered extensively for dynamic system modeling. Fuzzy, i.e., soft, model tree learning algorithms have already been introduced and used in the system identification and control domain, but their computational complexity is higher as compared to the crisp approaches.

Two intermediate goals comprise the main goal of this thesis. They concern modeling of single-output and multi-output systems, respectively. The first considers analyzing the similarities and differences between the two approaches for modeling dynamic systems, and their further development, i.e., introducing modifications or improvements to both approaches. We will first study, evaluate and analyze the similarities and differences between the crisp and fuzzy model tree learning algorithms introduced in both areas. The crisp model tree algorithms are fast but may produce inaccurate models when fitting smooth functions, while the fuzzy approaches are slower but more accurate.

Next, we will introduce modifications to a general-purpose crisp model tree learning algorithm in order to make it more accurate in both single-tree and ensemble settings. Also, we will introduce modifications to the more accurate fuzzy model tree learning algorithm in order to make it faster.

The second intermediate goal of this thesis is to study the multiple-output modeling problem, and propose solutions based on using multi-target model trees. The terminal nodes of the multi-target model trees include local linear models for all output variables of the system, which may be advantageous as the potential inter-dependence of the output variables can be utilized. Also, in the fuzzy model tree algorithm Lolimot, learning multi-target model trees is advantageous, since the evaluation of the intermediate models is performed simultaneously for all outputs. In more detail, the parallel simulation of the models for all output variables, performed during learning, is more sensitive to incorrect or imperfect models.

1.2 Methodology

First, we will define the problem of discrete-time modeling of dynamic systems as a static function approximation task, and as such will survey the machine learning methods appropriate for solving it. We will also survey relevant literature from the system identification and control community, mainly focusing on the multiple-model approaches and fuzzy modeling.

Second, we will study the applicability of crisp linear model trees, fuzzy linear model trees and ensembles of these two types of trees. We will also consider prediction of multiple dependent variables, i.e., targets. For multi-output dynamic systems we will build models which predict all outputs simultaneously. Ensembles of multi-target trees will also be considered.

Third, we will perform an empirical evaluation using benchmark machine learning datasets for the static case, and an empirical evaluation using standard measured and synthetic datasets from the field of non-linear system identification and control. For the latter, multi-output dynamic case studies will also be considered, and the multi-target model tree approaches evaluated. For the evaluation in the static case, the standard 10-fold cross-validation method will be used, while for the dynamic case, an evaluation based on training, validation and test sets will be utilized. In the latter case, it is not possible

to use cross-validation because of the temporal order of the data points that needs to be preserved. The performance results will be analyzed by using statistical tests, which will show which differences in performance are significant.

1.3 Contributions

The work presented in this thesis makes the following original contributions to the fields of machine learning and system identification:

- Design and implementation of novel model tree based approaches for modeling dynamic systems, based on and improving upon the M5' and Lolimot algorithms (Chapter 4):
  - Improved M5' algorithm for regression, which can now induce fuzzified and multi-target model trees.
  - Improved Lolimot algorithm for modeling dynamic systems, which now produces trees with similar predictive performance faster.
  - Algorithms that can induce ensembles of single and multi-target model trees by using the improved M5' and Lolimot algorithms.
- Empirical evaluation of the developed approaches on benchmark problems and case studies (Chapters 5 and 6):
  - Evaluation of the improved M5' and Lolimot algorithms (and ensembles based on these) on benchmark problems of single and multi-target static regression (Chapter 5).
  - Evaluation of all the above mentioned approaches (and a few other selected methods) on several case-studies of modeling dynamic systems (Chapter 6).

1.4 Structure of the Thesis

The introductory chapter describes general perspectives and topics considered in the thesis. It determines the goals of the research and the expected original contributions of the work. The rest of this chapter outlines the contents of the remaining chapters, which make up the thesis.

Chapter 2 presents system identification in discrete time, its objectives and the typical system identification loop. It also describes the external dynamics approach which allows a dynamic system identification problem to be transformed into a static regression problem. Finally, it outlines existing related work from the area of machine learning, suitable for solving regression problems.

Chapter 3 introduces tree-based methods for regression. It starts by describing the main components of tree learning algorithms. Then, it describes the details of the crisp model tree approach M5' and the fuzzy model tree approach named Lolimot. At the end, it outlines the limitations of each of the methods, which serve as a motivation for the work described in the following chapters.

Chapter 4 presents the novel approaches to modeling dynamic systems. It first introduces the modifications of the crisp M5' model tree algorithm that improve its accuracy for modeling dynamic systems and allow for multi-target prediction. Then the modifications

to the Lolimot algorithm that improve its learning time and modify its structure determination are described. Next, ensembles of model trees are introduced. Finally, the methods are illustrated on a static function approximation problem.

Chapter 5 presents the evaluation of the model tree ensemble approach based on the modified M5' algorithm described in Chapter 4 on the task of static regression. Both single-target and multi-target regression problems are considered. The evaluation is performed using benchmark machine learning regression datasets.

Chapter 6 reports the results of the evaluation of the approaches described in Chapter 4 on the task of modeling dynamic systems. It describes the dynamic system case studies, the preprocessing of the measured and synthetic data and the experimental results. The results reported consider both single-output and multi-output dynamic systems, by using single model trees and ensembles thereof. Also, a comparison to selected methods typically used for system identification is performed and reported.

Chapter 7 presents the conclusions and summary of the thesis, its original contributions, and gives some directions for further work.

Chapter 2

Background

Dynamic systems are systems whose responses change over time. The task of modeling dynamic systems is of high practical importance, because such systems are ubiquitous across all areas of life. The models allow for better understanding of dynamic systems, as well as their control, the latter being the focus of study in control engineering.

This thesis is positioned at the intersection of the fields of machine learning and automated modeling of dynamic systems. Machine learning is a sub-area of artificial intelligence, while automated modeling of dynamic systems is a sub-area of control engineering and system identification. Below we first discuss the broader range of system identification, which includes a complete description of the system identification loop. As this thesis considers a machine learning approach to system identification, in the remainder of this chapter we present existing machine learning approaches, which are used for regression. Finally, we discuss the application of machine learning methods to system identification.

2.1 System Identification

System identification is concerned with building models that closely describe the real world phenomena. In some cases it is possible to build the model using only thorough scientific understanding of the system. This means that the knowledge of the physical, chemical or other laws that govern the system may be enough to obtain a good model of it. This type of modeling is white box modeling, or first-principles modeling. On the other extreme, the task of modeling can be approached by using only measurements of some variables of interest. This is called black box, or data-driven modeling.

Objectives. The objectives of system identification are twofold (Billings, 2013). The first objective is building a model that would have good approximation and prediction properties, i.e., minimum prediction errors. Users who have this as their aim could find for example fuzzy logic approaches, neural networks, or related methods to be quite appropriate for achieving this. However, there is another, more analytical objective of system identification: the possibility to build a model that would reveal the components and behavior of the system. Having a model that reveals the behavior, users are also able to analyze the system by different means. One example is analyzing the system using different input signals, which may be in practice expensive to achieve, or even dangerous (e.g., nuclear systems).

The latter objective of interpretability is quite important for analyzing the obtained models. Other related objectives are obtaining insight into certain phenomena of the system, analyzing process behavior, controlling the process, or estimating some variables of the system that cannot be easily measured (Keesman, 2011). When the dominant objectives of the system identification are of this type, approaches such as neural networks

may not be appropriate. They would produce models which may have good predictive properties, however, the amount of parameters that are being estimated inside need not carry a direct relation to variables of the system (Billings, 2013). For such reasons, defining the purpose of the modeling procedure would also add bias to the selection of the most appropriate modeling technique.

Exact vs approximate models. The result of system identification is a mathematical model which represents an approximation of the real system. Due to the complexity of the system or the limited prior knowledge regarding the laws that govern it, one can only produce approximate models (Keesman, 2011). Even if the prior knowledge of the physical, chemical or biological laws that govern the system is complete, using the exact model would be too complex for particular applications. The look-up table models, which are quite popular in industry, are another example. They are simple, and only approximate models of the system.

The system identification loop. Guided by the intended purpose of the model and the prior knowledge of the system being modeled several decisions have to be made during the system identification procedure. The procedure itself can be divided into several steps, where these decisions are being undertaken. The steps of the system identification procedure can be summarized as:

- choice of model inputs,
- choice of model architecture,
- choice of dynamic representation,
- choice of model structure and complexity,
- choice of model parameters,
- model validation.

The steps are executed in the order given in Figure 2.1. The whole procedure is a loop (Nelles, 2001), since some of the steps need to be repeated, in order to improve the model of the system. In the remainder of this part, we outline some basic details of the system identification loop, which are also relevant to the thesis.

The *choice of model inputs* determines which variables would be used in the modeling procedure as inputs. Additionally, it determines what kind of excitation signal would be used for each input (sinusoidal, step-like, etc). The input signals have to be chosen in such a manner that they excite all of the system modes, enough for determination of the model (Ljung, 1987).

This step is one of the more important steps in the process. It requires knowledge of the purpose of the modeling, as well as prior knowledge of the system that is being modeled. The set of input variables could be chosen by using unsupervised or supervised input selection, trying all input combinations, or other pruning techniques. It is worth noting that, in spite of its importance for a black-box modeling technique, the choice of excitation signals is usually limited, due to its application specific nature.

The decision of which *model architecture* would be used is the hardest and at the same time subjective. Different criteria play a role in the selection of one model architecture over another. Some of them include: intended use of the model, dimensionality of the problem, offline or online learning and experience of the user.

The *choice of dynamic representation* is also influenced by the intended use of the model. The most frequently used is the external dynamics approach, however the internal

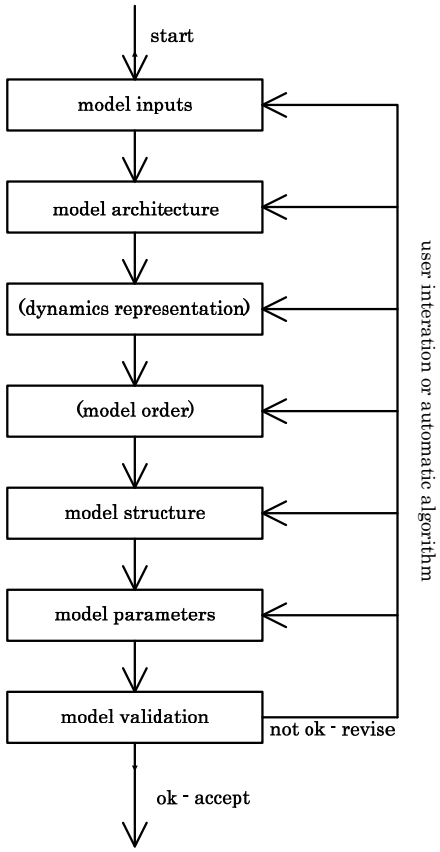


Figure 2.1: The system identification loop according to Nelles (2001).

dynamics may also be used, mostly in cases when there is little knowledge of the underlying system (Nelles, 2001).

The *choice of model order* is influenced by the prior knowledge of the system being modeled. However, many practical applications make use of the trial-and-error approach. It is worth noting that considering higher dynamic order (lag) leads to a loss in static approximation accuracy (Nelles, 2001).

The *choice of model structure* in the dynamic case is concerned with selecting a subset of the available variables to be used as regressors. It can be performed automatically (for example with the Orthogonal Least Squares approach) or by using non-automatized trial-and-error approaches. The algorithms used in this thesis are able to automatically determine the model structure, i.e., the regressors to use in the linear models.

The *choice of model parameters* is performed by using different linear or nonlinear optimization techniques. The literature and standard toolboxes offer many different approaches. This makes the usage of various techniques easy and practical, i.e., the user can apply the optimization procedure in a black-box fashion.

The *model validation* determines whether the obtained model is acceptable for its intended purpose. Testing the model using an unseen test dataset could be considered as a first step. In the case there is lack of data, i.e., all the data were already used for training, the model could be tested using simple (synthetic) input signals, such as step-like signals. This should provide some insight into the model, even in the case there is no data to compare the predictions to.

Given the importance of the purpose of modeling, obtaining a model may be only the first stage of a multi-stage procedure. For example, the next stage could utilize the model to design a controller or a fault-detection system. In this case, the criterion for validation would be the performance of the derived controller, or fault-detection system.

2.1.1 Discrete-time vs Continuous-time Modeling

Dynamic systems can be modeled in continuous time with systems of ordinary differential equations, describing the rate of change for each of the system variables. They can also be modeled in discrete time, by using difference equations that describe the state of the system at (a discrete) time point k as a function of previous system states and inputs.

2.1.1.1 Continuous-time modeling

In continuous time, the models of dynamic systems take the form of ordinary differential equations (ODEs) or partial differential equations (PDEs). The rate of change of state variable of the system is expressed as a function of the current state of the system, as well as the values of input (exogenous) variables.

2.1.1.2 Discrete-time modeling

In a state-space system, the next state of the system is determined by the state equation, in terms of its current state and inputs. The system output is determined by the output equation, in terms of a combination of the current system state and the current system input. The measured variables are the input (u) and output variables (y).

In discrete time, the task is to obtain difference (recurrence) equations, by using the measurements for the input and output variables. These equations would describe the current outputs of the system using past values of the input and output variables. Through the *external dynamics approach* (Nelles, 2001), the modeling problem can be reformulated as a regression task. The value of the output variable(s) at time instant k, $y(k)$, needs to

be predicted from the lagged values of the input and output variable(s), $u(k-1), u(k-2), .., u(k-n_u), y(k-1), y(k-2), .., y(k-n_y)$ using a static function approximator. The values n_u and n_y are the lags of the input and output variables, respectively.

It is worth noting that certain differences exist between continuous-time modeling and discrete-time modeling. First, the number of terms in the equations of continuous-time models is smaller. For example, a first order derivative term may require up to three lagged values of the corresponding variable (Billings, 2013). This translates to a larger number of parameters that have to be estimated in the discrete-time model. Second, discrete-time modeling has a requirement that a sampling time needs to be defined. Moreover, the choice of sampling time could also affect the success and efficiency of the discrete-time modeling procedure. Also, in both cases the data collection stage inevitably involves data sampling, which is a discrete process.

2.1.2 System Identification in Discrete Time

This thesis addresses the task of discrete-time modeling of nonlinear dynamic systems. As mentioned above, two types of variables are used in modeling, output and input (exogenous) variables, denoted by y and u, respectively. Using the external dynamics approach, the task of empirical modeling of a dynamic system can be formulated as a regression problem of finding a difference equation that fits an observed behavior of the system.

More precisely, to model a system described by y and u, we need to formulate a difference equation that expresses the value of the output variable y at a given time point k as a function of past output and input variables (y and u). The transformation creates a new vector of features, which is composed of the lagged values of the input variable u and output variable y. Typically, up to n_u and n_y time points in the immediate past, with respect to k, are considered, for the variables u and y respectively. At time point k, the dynamic system is thus represented by the vector of features $\mathbf{x}(k)$

$$\mathbf{x}(k) = [u(k-1), u(k-2), .., u(k-n_u), y(k-1), y(k-2), .., y(k-n_y)]^T \tag{2.1}$$

where (n_u, n_y) are the orders (lags) of the system[1]. The model of the system is a difference equation that describes the output of the system at (a discrete) time point k, $y(k)$ as a function of the previous system states and inputs (i.e., $\mathbf{x}(k)$). The corresponding regression problem is to train a nonlinear function approximator $f(.)$, s.t. $y(k) = f(\mathbf{x}(k))$, from a table of data generated from an observed behavior in the manner described above.

The task of discrete-time modeling of nonlinear dynamic systems from measured data can be approached using different modeling techniques. Over the last few decades, numerous different methods have emerged. The earlier approaches include for example the block-oriented Hammerstein and Wiener systems (Giri & Bai, 2010) and the semi-parametric Volterra method (Haber & Unbehauen, 1990). More recent approaches include the widely used basis-function approaches of artificial neural networks (Nelles, 2001) and fuzzy modeling, as well as the nonparametric approaches of kernel methods (Cristianini & Shawe-Taylor, 2000) and Gaussian Process models (Rasmussen & Williams, 2006), to list just a few.

As discussed in Section 2.1, the last step of the system identification loop is a validation of the dynamic system's model. The validation is carried out according to the purpose of the model and often requires a stringent and purpose-specific procedure. When the purpose is good one-step-ahead prediction performance, as shown in Figure 2.2 (a), the predicted values for the system variable are compared to the measured values. On the other hand, if

[1] In the thesis we consider only the special case of $n_u = n_y$. Using this simplification, we consider an identical order for all input and output variables.

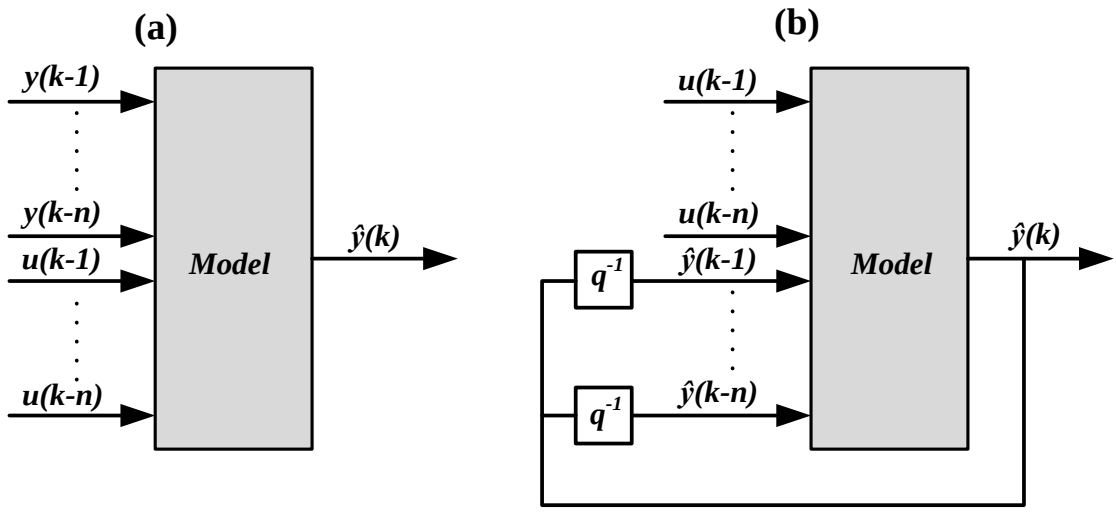


Figure 2.2: A model used for prediction (a) or simulation (b). (q^{-1} is the backshift operator)

the aim is simulation, illustrated in Figure 2.2 (b), there is one substantial difference: the one-step-ahead model predictions are fed back to the model to produce predictions for the more distant future.

While the first step of one-step-ahead prediction and simulation is the same, in simulation, the predicted value of the system variable y at time k (i.e., $\hat{y}(k)$) is fed back as input to the model, instead of a measured value ($y(k)$) at time $k+1$. Due to the realistic possibility of error accumulation in the case of an inaccurate model, divergence of the simulation predictions from the measured values may occur as we move further into the future. The cumulative error of simulation is referred to as the *output error*, while in the case of one-step-prediction the error is referred to as *prediction error*. While most approaches to solving this task try to minimize the one-step prediction error, the learned models are typically evaluated in terms of their simulation (output) error.

2.2 Machine Learning Approaches to Regression

A standard task in machine learning is learning predictive models. Predictive models predict the value (or values) of a dependent variable from the values of independent variables (predictors). The predictive modeling tasks are divided based on the type of the target variable into classification (discrete target variable) and regression (continuous target variable). Among the most popular predictive modeling approaches for regression are artificial neural networks (ANN), kernel approaches such as support vector regression (SVR) and tree-based approaches. In the remainder of this section we present some details of these methods.

The artificial neural networks (ANNs) (Nelles, 2001; Narendra & Parthasarathy, 1990), which can be seen as universal approximators, are very powerful and flexible methods for regression, which are also universal function approximators. They typically consist of many simple computational elements, arranged in layers and operating in parallel. The ANN methods are determined by their network architecture, node characteristics and learning procedures. Different architectures of neural networks exist, the most common ones being multilayer perceptron (MLP) and radial basis function (RBF) networks. They were invented as early as 1943 (McCulloch & Pitts, 1943), uniting the studies of neurophysiology

and mathematical logic. However, the most noticeable contribution, the backpropagation of errors procedure (Rumelhart, Hintont, & Williams, 1986) was introduced later, in 1986.

The ANNs, which can also be seen as a basis-function approach (Nelles, 2001), work by fixing the number of basis functions, and allowing them to be adaptive. The basis functions include several parameters, which are adapted during the learning process (Bishop et al., 2006).

Typically, the basis functions in ANNs are implemented in nodes, which are grouped into layers. In this architecture, the nodes that produce the output of the model are grouped in an output layer, while the network can include a group of hidden nodes, forming the hidden layer. The parameters of each of the basis functions are adapted during the training process. The most frequently used learning algorithm is the backpropagation of the error, performed by starting from the output layer, and working back towards the hidden and input layers. However, in spite of the advantages of popular ANN approaches, their main disadvantages are the lack of transparency and curse of dimensionality (Ažman & Kocijan, 2011).

The support vector machines (SVM) for classification and their regression variant of support vector regression (SVR) utilize the concept of kernels, formulated as an inner product in a feature space. The concept of kernels was introduced as early as 1964 (Aizerman, Braverman, & Rozoner, 1964), but it found its place in the machine learning domain later, after the work of Boser, Guyon, and Vapnik (1992).

The SVR addresses the curse of dimensionality issue by defining basis functions that are centred in the training points, and then selecting a subset of these, named support vectors, during training. In other words, the model is expressed in terms of only a few support vectors, and is able to approximate nonlinear functions with the help of the kernel mapping.

The *tree-based approaches* are quite popular and the machine learning literature provides many different algorithms for learning classification and regression trees (Breiman, Friedman, Olshen, & Stone, 1984), as well as the special case of model trees ((Karalič, 1992), the M5' algorithm (Wang & Witten, 1997; Frank, Wang, Inglis, Holmes, & Witten, 1998), the HTL algorithm (Torgo, 1997), and others (Loh, 2002; Dobra & Gehrke, 2002; Malerba, Esposito, Ceci, & Appice, 2004; Gama, 2004)). The model trees are comprised of inner nodes with splits, and terminal nodes, which contain a local model (either constant, linear, polynomial or of some other more complex type). Linear model trees with axis-orthogonal splits are defined as: A model tree consists of either a) a split node with a test of the form $variable \leq threshold$ which creates a binary partition of the input space and has left and right offspring nodes or b) a single terminal node for which a local linear model is defined.

Tree learning algorithms are robust and tend to scale well to large predictive modeling problems. The algorithms apply the divide-and-conquer principle, by splitting the available learning data into smaller subsets as the tree is constructed. Thus, the potentially complex optimization problem is broken down to several simpler optimization subproblems. Each optimization subproblem uses a proper subset of the whole set of training data, so the learning procedure is simplified. This gives the tree learning algorithms the ability to efficiently handle a large number of data points (i.e., instances).

The issue of overfitting to noise in decision trees and tree ensembles is controlled by the depth of the individual tree or the trees in the ensemble (Segal, 2004). Larger trees are more sensitive to noise and prone to overfitting, while smaller trees are less sensitive and less prone to overfitting. Tree pruning procedures can be used to reduce the depth of an overly large tree, and control the overfitting to noise.

2.3 System Identification with Machine Learning: Prior Work

The procedure of system identification concerns building models of real-world phenomena, typically by defining a mapping between a set of variables called inputs, and another set, called outputs. Several approaches exist for achieving this goal, which have been introduced in different domains. It is worth noting that while the categorization of existing approaches for nonlinear system identification ("Nonlinar System Identification," 2014) contains the traditional block-oriented Hammerstein and Wiener systems (Giri & Bai, 2010), and the semi-parametric Volterra method (Haber & Unbehauen, 1990), the approaches appearing more recently are denoted using the phrase "neural network models". It is typical to use the phrase "neural network models" for all machine learning approaches, i.e., approaches which learn from data. This is due to the early success of the neural networks for system identification. Also, the naming of "neuro-fuzzy" model class suffers from similar shortcomings. In its early period, the neuro-fuzzy models were approaches which applied neural networks techniques for learning parameters (and/or structure) of fuzzy models. At present, however, this model class has been extended to different kinds of machine learning methods which learn the parameters and/or structure of fuzzy models (Nelles, 2001).

In the remainder of this section we first introduce the existing machine learning approaches used for system identification, categorized into two categories, depending on the models they produce. Then, we outline one important challenge for the methods: assuming that the intended usage of the dynamic system models is simulation, we present the problem of optimizing the simulation error during learning.

2.3.1 One Global Model vs Multiple Model Approaches

This part presents a classification of the machine learning approaches according to the type of model they produce. Some approaches learn one global model describing the whole system, while others learn multiple models. The following paragraphs present the methods belonging to the two categories.

Methods that build one global model include for example the well-known artificial neural networks (Narendra & Parthasarathy, 1990), Gaussian Process models (Rasmussen & Williams, 2006), and support vector regression (Cristianini & Shawe-Taylor, 2000). They learn one global model by using a nonlinear optimization procedure on all training (identification) data in a single optimization task. The learned model is valid in the whole operating region.

Gaussian Process models are nonparametric, probabilistic black-box models that have been used for modeling dynamic systems (Rasmussen & Williams, 2006). One of their advantages is the measure of confidence for the predictions they provide, which helps in assessing the quality of the model prediction at each point. This approach is related to support vector machines and especially to relevance vector machines (Rasmussen & Williams, 2006).

The multiple model approaches build several local models, each of them valid in a subregion of the whole operating region. They are also referred to as local model networks (Murray-Smith & Johansen, 1997). They include neuro-fuzzy approaches like the Adaptive Neuro Fuzzy Inference System – ANFIS (Jang, Sun, & Mizutani, 1997), Local Linear Model Trees – Lolimot (Nelles, 2001), and the operating regime approach (Johansen & Foss, 1997).

The ANFIS method is a hybrid neuro-fuzzy approach, which builds a Takagi-Sugeno fuzzy model (Takagi & Sugeno, 1985). It is worth noting that this approach differs from neural networks, because it has different types of nodes and it utilizes a hybrid learning approach. Also, the resulting model is a set of fuzzy rules, i.e., a Takagi-Sugeno fuzzy model. ANFIS uses a hybrid learning rule that combines the backpropagation gradient

descent and the linear least-squares optimization method, for its parameter estimation. The structure identification task, i.e., the determination of the number of fuzzy rules and initial positioning of the fuzzy rule centers can be handled by using different methods: grid partitioning of the instance space, fuzzy clustering, or a tree-based approach (Jang, 1994). The last decision is typically left to the user.

The Local Linear Model Trees (Lolimot) (Nelles, 2001) method is a multiple model approach which builds a fuzzy model tree. It solves the structure identification and parameter estimation problems in an integrated, iterative procedure. In each iteration, the method adds one local model to the tree structure and calculates the parameters of the model using local parameter estimation. It has been successfully used for identification of dynamic systems (Nelles, 2001). More details about the Lolimot method are presented in the following chapter of the thesis.

All methods mentioned so far, i.e., ANNs, ANFIS and Lolimot, suffer from the curse of dimensionality as the number of input dimension gets larger. This is especially problematic for the ANFIS method in case the structure determination is performed by using grid partitioning. The number of parameters that it needs to estimate is in this case proportional to p^{n_u} (where n_u is the number of input variables and p is the number of membership functions assigned to each variable).

2.3.2 Optimization of the Output Error

Most approaches to modeling dynamic systems optimize the prediction error during learning, both for the structure determination and the parameter estimation. However, the validation of the learned model is typically performed by simulation. This presents a challenge and raises the question whether it is possible to directly optimize the output error while learning, instead of optimizing the prediction error. In control engineering, the optimization of the prediction error is also known as a series-parallel identification structure, while the optimization of the output error as a parallel identification structure. For example, the related work of Connally, Li, and Irwin (2007) investigates the differences between series-parallel and parallel identification structures for training neural networks. Their work tries to combine the optimization of the prediction and output error in one training algorithm, in order to provide a more accurate neural network model.

Several other related works (L. Piroddi & Spinelli, 2003; Luigi Piroddi, 2008; Nelles, 2001; Kocijan & Petelin, 2011) deal with the parallel identification structure. The works of Nelles (1995, 2001) conclude that a decrease of the prediction error does not necessarily lead to a decrease in the output error of the model, when using neural-networks as models. Similarly, Kocijan and Petelin (2011) deal with the same question in the context of Gaussian Process models. They conclude that the direct optimization of output error is a much harder task as compared to the optimization of prediction error, since the optimization surface in the former case contains many local optima and its contour plots depict quite irregular shapes.

However, some nonlinear identification methods exist, which make use of the output error for model structure selection. An example is the Lolimot method (Nelles, 2001), which iteratively adds complexity to a tree structure. In each iteration, the method solves the parameter optimization problem using least squares estimation, evaluates the intermediate model using simulation, and tries to improve the structure by adding one more node to the tree. The author concludes that the structure search, a nonlinear optimization problem solved by a greedy hill-climbing approach, could benefit from directly estimating the simulation error. This approach is possible because: a) the iterative nature of the approach means that after each iteration an intermediate solution, i.e., a fuzzy model tree, is ready to be used; b) the number of iterations, or total number of nodes in the tree is

typically not large, so the time consuming evaluation of the output error on the whole training set does not increase the overall time complexity substantially.

Chapter 3

Tree-based Methods

This chapter first introduces the model trees, and the differing terminologies of machine learning and system identification. Then it describes the main components (building blocks) of model tree learning algorithms. This is followed by an introduction of the M5' algorithm, and its main components. In the following, the Lolimot model-tree algorithm for dynamic system identification is presented. Finally, the potential limitations of the existing mode tree algorithms for modeling dynamic systems are stated. These serve as a motivation for performing modifications and improvements to the algorithms.

3.1 Introduction

A tree learning method from machine learning produces binary trees with two types of nodes: inner nodes and terminal nodes, cf. Figure 3.1 (a). The inner nodes of the trees contain a split, of a certain form. The type of splits used could range from axis-orthogonal (single split variable), axis-oblique, functional or other more complex split types. The terminal nodes of the trees contain a local model, which could be constant, linear, polynomial or some other more complex type. This thesis is concerned with trees which have axis-orthogonal splits in the inner nodes and linear models in the terminal nodes. Such trees are named *model trees*.

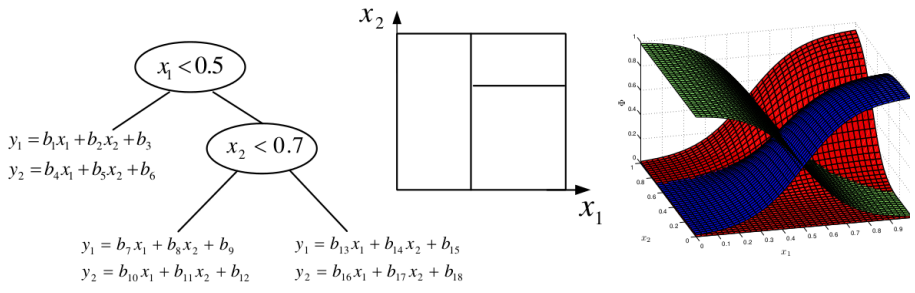
A set of splits of one model tree, followed from the root down to a terminal node, defines one partition. The set of all partitions that a tree defines is named partitioning. In the machine learning literature, most of the tree-based approaches build crisp trees, i.e., trees with crisp or hard splits. A data point (i.e., an instance) belongs to exactly one partition, i.e., exactly one linear model is associated to a data point. For each of the partitions, or terminal nodes, the linear model learned is also named local model (LM).

However, by using fuzzy sets and fuzzy logic, several researchers have also defined fuzzy trees. They differ in the fact that the splits use fuzzy weighting: a data point belongs to the left subtree with a certain weight, and to the right subtree with a different weight. This also means that now all of the local models are associated with a data point, however, each one with a different weight (cf. the fuzzy weighting in Figure 3.1 (a)).

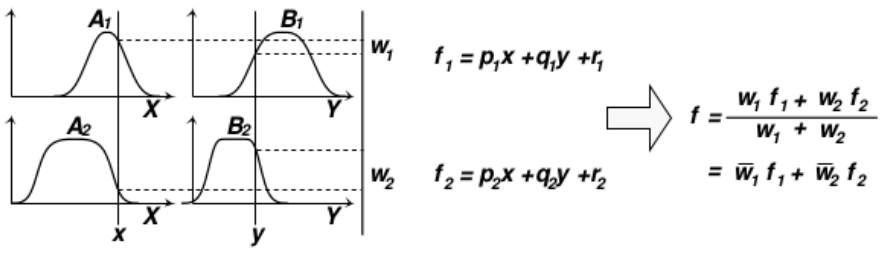
An example of a method used in control engineering, which produces fuzzy model trees is the Local Linear Model Trees method (Lolimot). The model is built with the help of the tree-based partitioning of the operating region, but at the end of the learning procedure, it is represented as a Takagi-Sugeno (TS) fuzzy model (Takagi & Sugeno, 1985; Jang et al., 1997). However, the form of the Takagi-Sugeno fuzzy model and the fuzzy model tree described above are equivalent.

A Takagi-Sugeno fuzzy model, shown in Figure 3.1 (b) is defined by a set of rules. Each rule is comprised of a left-hand side (antecedent) which is a fuzzy partition, and a right-

hand side (consequent) which is a crisp linear model. The model tree built by the Lolimot algorithm is converted to a set of TS rules. The conversion procedure considers each of the partitions defined by the model tree partitioning. For each partition (hyperrectangle), a fuzzy membership function is placed in the center of the partition, which comprises the antecedent part of the TS rule. The consequent part, which does not include fuzzy components, is a local linear model associated with that rule.



(a)



(b)

Figure 3.1: (a) A fuzzy model tree; (b) A Takagi-Sugeno model.

3.2 Model Tree Learning Algorithms

This part gives an overview of the existing model tree learning algorithms. It starts with a description of a general model tree algorithm and its main features. Then, each of the features of the algorithm is discussed in more detail. The discussion focuses on the operation of model tree learning algorithms, governed by the top-down induction of decision trees (TDIDT) principle (Blockeel & De Raedt, 1998). Finally, to put the Lolimot method in context, this part discusses an alternative to the recursive top-down induction principle, along with its motivation.

Tree growing phase. The tree growing phase builds the initial tree structure. The top-down learning approach starts by defining the root node of the tree. Each node of the tree is then expanded until some of the stopping criteria are met, thus building the tree structure. The expansion of a terminal tree node is performed by replacing it with a split node and its two immediate descendant nodes.

When a terminal node is considered for expansion, a *split selection* procedure selects a split to be added to the tree, among several *candidate splits*. This procedure is the most important part of a TDIDT algorithm. The existing approaches use different strategies for implementation of the split selection heuristic. The work of Vens and Blockeel (2006) categorizes the model tree algorithms based on the split selection heuristic they implement. To be more exact, the authors perform the categorization based on the the linear regression

that is being used for split selection: whether it is performed or not, and what kind of linear regression. The four categories are shown in Table 3.1. The split selection which does not perform regression, and instead utilizes a fast heuristic, has a linear complexity in the number of splitting attributes. However, the authors claim that a split selection procedure which performs simple regression using only the split attribute as a regressor can also be implemented as a linear procedure in the number of split attributes, i.e., quite fast. The last two alternatives in Table 3.1 perform a more thorough investigation for the optimal split, but are slower.

Table 3.1: A categorization of the model tree algorithms based on their split selection procedure, based on the work of Vens and Blockeel (2006).

Regression in split sel.heuristic	complexity	example methods
no regression in spl.heuristic	linear	M5' (Wang & Witten, 1997)
simple regression using only the split attribute	linear	MAUVE (Vens & Blockeel, 2006)
separate simple regressions using one attribute	quadratic	SMOTI (Malerba, Esposito, Ceci, & Appice, 2004) Treed regr. (Alexander & Grimshaw, 1996)
multiple regression using all attributes	cubic	RETIS (Karalič, 1992) Lolimot (Nelles, 1999)

These categories of heuristics categorize the available methods by the regression performed only during split selection. The regression that a model tree algorithm performs to learn the LMs is considered as a separate issue. It is worth noting that the methods from the last three categories shown in Table 3.1 are also typically named "look-ahead" approaches, since they build linear models and evaluate their accuracy in the split selection step. The growing phase is stopped for a certain terminal node, when one of the stopping criteria is satisfied.

One subproblem of the split selection problem, is the *split cut-point optimization*, or split threshold optimization procedure. It is concerned with determining the optimal split point, given a split variable A. The prior text has provided information regarding the different evaluation functions. These are used to select one split out of several candidate splits, and as discussed, can either be fast heuristics which do not perform regression, or different type of slower look-ahead heuristics, which perform regression.

Different methods exist for split cut-point optimization, and they can be divided into data-oriented and interval approaches. The data-oriented approaches look at the values of the variable A of the data points falling in that partition, sort the values, and consider a subset of the unique midpoints, cf. Table 3.2. This type of implementation is typical for methods from the machine learning domain. On the other hand, the interval implementations are not concerned with the distribution of the data points for the partition, hence this alternative is not data-oriented. Instead, it only determines the extreme (minimal and maximal) data points, and places the candidate cut-points at uniform locations in that interval. This is summarized in Table 3.2. Also, the number of candidate cut-points considered for each variable has a tremendous impact on the computation time required for a model tree algorithm with a look-ahead split evaluation.

Tree pruning. It is well known that overly large trees are prone to overfitting. Tree pruning is a method that handles overfitting by removing tree nodes which may deteriorate the performance of the tree. Depending on the particular model tree algorithm, the tree may only be pre-pruned, or it may also include a post-pruning step. In the former case, the

Table 3.2: Candidate cut-point determination for a split attribute A. A_{min} and A_{max} denote the minimal and maximal value of this attribute in the set of data points.

type	Set of candidate cut-points	used in
data-orientated	all unique midpoints between the values of the data points	M5' (Wang & Witten, 1997)
	every n'-th unique midpoint between the values of the data points	MT-SMOTI (Appice & Džeroski, 2007)
interval	half point of the interval $[A_{min}, A_{max}]$	Lolimot (Nelles, 1999)
	5-20 uniformly dist.points in the interval $[A_{min}, A_{max}]$	Reg.Decomposition (Johansen & Foss, 1995)

tree growing is stopped by evaluating some stopping criteria. In the latter, a post-pruning phase is included after the initial and overly large tree is built.

Fuzzy trees. The commonly used model tree algorithms (Quinlan, 1992; Dobra & Gehrke, 2002; Malerba et al., 2004; Karalič, 1992) utilize the recursive divide-and-conquer approach and produce only hard or crisp splits. Another research direction considers trees with soft or fuzzy splits (Marsala, 2009; Lemos, Caminhas, & Gomide, 2011). The fuzzy trees, mainly developed for solving regression problems, produce models which fit smooth regression surfaces better. The discontinuities produced by the crisp tree building algorithms are smoothed, which results in more accurate models. The machine learning literature provides many different approaches for learning fuzzy regression trees (Suarez & Lutsko, 1999; Olaru & Wehenkel, 2003), i.e., trees with constant local models, and a few approaches for learning fuzzy linear model trees (Lemos et al., 2011), which are equivalent to a Takagi-Sugeno model.

Some of the approaches directly learn trees with fuzzy splits, while others learn crisp splits and convert them to fuzzy afterwards. In other words, the introduction of fuzzy splits in the tree learning algorithm may be performed

- within with the look-ahead split selection function, or,
- after the tree growing stage.

In the first case, the fuzzy membership values are typically used in the local model estimation (Suarez & Lutsko, 1999), which means that a weighted least squares regression is performed. This in turn increases the complexity because the evaluation of the model tree for a single data point requires that predictions from all local models be calculated, as well as each of their associated weights. Also, the model tree algorithm could aim for a more accurate model, and the parameters of the fuzzy splits can be optimized by an additional, usually expensive, step (Olaru & Wehenkel, 2003) performed at the end.

In general, the fuzzy tree methods are more computationally complex as the data points from neighboring partitions also influence the local model estimation in a given partition of the input space. This means that the parameters of the local models for each partition are calculated using all training data, where the data points that are part of the partition receive the highest weights. This procedure is clearly different from the more efficient top-down induction of decision trees (TDIDT) approach, discussed earlier.

Iterative learning. Apart from the popular and efficient TDIDT approach, there is an alternative iterative approach to learning trees, used mostly for fuzzy trees. It is motivated by the fact that the prediction of a fuzzy model tree is calculated using all local model predictions, so it would be more appropriate to evaluate the whole tree, rather than each local model in isolation. The evaluation of the whole tree would assess the interactions

between the local models, and their effect on the final model tree prediction. Another reason for the iterative learning are potential constraints that the user might pose on the model (Kocev, Struyf, & Džeroski, 2006) before learning. These would also require an evaluation of the model as a whole.

The Lolimot algorithm is a model tree learner which operates iteratively, and which assesses the overall performance of the tree during learning. In each iteration, the split node to be added is determined by using the overall (or global) model performance (Nelles, 2001).

Algorithms such as Lolimot, that build model trees in an iterative fashion, are also able to minimize output error when modeling dynamic systems (Nelles, 1999). The simulation procedure for evaluating a model of a dynamic system requires the complete model to be known. The complete, but intermediate, models are available at the end of each iteration of the iterative learning method Lolimot.

Multi-target trees. Multi-target trees are decision trees, where the terminal nodes contain models for several target variables. Such trees are used when the modeling problem contains several dependent variables. They utilize the potential inter-dependence between the different target variables, and provide models with smaller complexity, as compared to modeling each target variable separately. In the case of model trees, the terminal nodes contain local models for several numeric variables (Appice & Džeroski, 2007), an approach also introduced for the fuzzy model trees (Nelles, 1999). However, the decision trees can also contain local models for several discrete variables, or even a set of discrete variables organized in a hierarchy (Aleksovski, Kocev, & Džeroski, 2009).

3.3 The M5' Model Tree Learning Algorithm

This part introduces the M5' algorithm (Quinlan, 1992; Wang & Witten, 1997) as implemented in the WEKA (Hall et al., 2009) framework. It presents the pseudocode and analyzes some of its aspects. First it discusses its tree growing procedure, introducing the fast variance reduction split heuristic and the pre-pruning criteria. This is followed by a description of the post-pruning phase, which reduces the size of an overly grown tree. Finally, it discusses the approach taken for handling discrete attributes, which may appear in the dataset.

3.3.1 Tree Growing Phase

Tree growing, shown in Algorithm 3.1, is a recursive procedure that generates the initial structure of the tree. The procedure consists of determining whether the tree node should be a split (inner) node or a terminal node containing a linear model. If a split node is created, the procedure continues recursively for the examples sorted down each of the two branches created by the split.

The decision to create a terminal node instead of a split node (i.e., to perform pre-pruning) is taken when one of the two stopping criteria are met. The first criterion tests if the number of training points in the current node is smaller than the value of the minimal number of instances parameter (n_{min}). The second criterion stops tree growing when the standard deviation of the target attribute on the data points falling in the current node is smaller than 5% of its standard deviation on the whole training set.

The selection of the split parameters (the feature attribute to split on and the cut-point) is guided by the standard deviation reduction (SDR) heuristic, shown in Eq. (3.1) below. Normally the split with the highest reduction in the standard deviation is chosen.

As previously noted, the standard deviation reduction (SDR) heuristic is used to evaluate all possible split cut-points. The feature attribute (A) and cut-point (c) combination in the test $[A < c]$ which maximizes the SDR heuristic is selected and used as a split at the current tree node. The SDR heuristic score is calculated as:

$$SDR = \sigma_D^2 - \frac{|D_l|}{|D|}\sigma_{D_l}^2 - \frac{|D_r|}{|D|}\sigma_{D_r}^2 \tag{3.1}$$

where D is the set of data points falling in the current tree node, D_l and D_r are the two subsets of data points corresponding to the left and right branches of the split. σ_D^2 denotes the standard deviation of the target attribute in the set D, while σ_{D_l} and σ_{D_r} denote the standard deviations of the target attribute in the sets D_l and D_r, respectively.

Algorithm 3.1: Pseudocode for the tree growing phase of M5'.

Algorithm Build_tree(D)
- **Data**: D - a training set
- **Result**: T - a tree
- **if** $|D| < n_{min}$ **then**
  - Return a terminal node
- **end**
- **if** *standard deviation stopping criterion is satisfied* **then**
  - Return a terminal node
- **end**
- Let $\{A_1, A_2, .., A_p\}$ be a random subset of feature attributes
- Initialize s_{best}
- **for** $k = 1, .., p$ **do**
  - Let split $s^*[A_k < c] = \underset{s}{argmax}(SDR(s[A_k < c]))$
  - **if** $SDR(s^*) < SDR(s_{best})$ **then** $s_{best} = s^*$
- **end**
- Split set D into subsets D_l and D_r based on split s_{best}
- Let $T_l =$ **Build_tree**(D_l)
- Let $T_r =$ **Build_tree**(D_r)
- Return a tree with a split node s_{best} and subtrees T_l and T_r

3.3.2 Tree Post-pruning Phase

The tree pruning, shown in Algorithm 3.2, is a method that handles overfitting by removing tree nodes which may deteriorate the performance of the tree. The post-pruning procedure is performed after the initial tree structure has been built. It takes into consideration the prediction error of the local models, as well as the prediction error of whole subtrees. The pruning procedure operates in a bottom up fashion: it starts by considering the terminal nodes for pruning and continues towards the root of the tree.

As a first step of the procedure, linear models are estimated (constructed) in all nodes of the tree, by using least squares linear regression. The default operation of M5' is to use a feature selection scheme: The local model is built only by using features found in tests of split nodes below the current tree node. This thesis evaluates M5' with the feature selection procedure, as well as M5' where this procedure is turned off. Also, the least squares estimation of the local linear model includes an attribute removal part: Features with small effect are dropped from the linear model (Wang & Witten, 1997).

After the estimation of linear models, the bottom-up pruning procedure evaluates whether to prune each tree node. It compares the accuracy of the linear model learned at the node to the accuracy of the subtree rooted at the node. A decision to prune (replace the subtree rooted at that node with a terminal node) is made only if the accuracy of the subtree is smaller than the accuracy of the linear model.

The M5' algorithm and its WEKA implementation also contain a procedure which modifies the coefficients of the linear models, in order to improve the accuracy of the model tree. This procedure, named smoothing, uses the coefficients of the local models learned in inner tree nodes, to smooth the local model prediction of a terminal node. The details of this procedure can be found in the following chapter, Subsection 4.1.1.1.

Algorithm 3.2: Pseudocode for the tree pruning phase of M5'.

```
Algorithm Prune(T)
    Data: T - a model tree
    Result: pruned model tree
    if root of T is a split node then
        Prune( T → left )
        Prune( T → right )
        Learn a linear model for the root node of T
        Calculate the error of the local linear model e_LM
        Let e_ST=Subtree_error(T)
        if e_ST > e_LM then
            Convert root of T to a terminal node
        end
    end
    Return T
Algorithm Subtree_error(T)
    Data: T - a model tree
    Result: numeric value
    if root of T is a split node then
        Let T_l=T → left
        Let T_r=T → right
        Let D=T → examples
        Let D_l=T_l → examples
        Let D_r=T_r → examples
        Return ( |D_l| * Subtree_error(T_l) + |D_r| * Subtree_error(T_r) ) / |D|
    else
        Return the error of the linear model in root of T
    end
```

3.3.3 Handling Discrete Attributes

The discrete, or nominal attributes in the dataset are transformed into several binary attributes, using the approach of Breiman et al. (1984). For each of the j possible values of a discrete attribute, the average of the target is computed, using the training examples. The averages are used to sort the j values. Based on this ordering, $j-1$ binary attributes are formed.

For example, assume that the nominal attribute A has three possible values v_1, v_2, v_3. Also, assume that the ordering based on the average of the target is v_3, v_1, v_2. The two

new attributes that this procedure creates are: $A' : A = v_2$ and $A'' : A = v_1 \vee v_2$. Details of the procedure are presented by Breiman et al. (1984), Wang and Witten (1997).

3.4 Lolimot

Algorithm 3.3: Pseudocode for the Lolimot method.

Algorithm Build_tree(D_{learn}, D_{sim})
- **Data**: data set used for learning D_{learn}, data set used for simulation D_{sim}
- **Result**: model tree T
- Create a root node for the tree and estimate local model parameters using data D_{learn}
- Evaluate the model tree: (e, t) = **Evaluate**(T, D_{learn}, D_{sim})
- **while** *the maximal number of LMs is not reached and the model error is above the threshold* **do**
  - Select the terminal node t for splitting
  - Create the set S of candidate splits using terminal node t (Eq. (3.4))
  - **for** *all candidate splits s in S* **do**
    - Replace the terminal node t with a split node t'_s and split s
    - Create two terminal nodes t'_1 and t'_2 as descendants of t'_s
    - Estimate local model parameters for t'_1 and t'_2 by weighted linear regression using D_{learn}
    - Evaluate the model tree: (e, t) = **Evaluate**(T, D_{learn}, D_{sim})
  - **end**
  - Select and keep the candidate split s which produces lowest overall error e_s
- **end**
- Let i_{AIC} be the iteration with the smallest AIC value (Eq. (3.15))
- Return the model tree T from iteration i_{AIC}

Algorithm Evaluate(T, D_{learn}, D_{sim})
- **Data**: model tree T, data set used for learning D_{learn}, data set used for simulation D_{sim}
- **Result**: squared error of the model tree e, terminal node t with largest error
- **if** D_{learn} *are data from a dynamic system* **then**
  - Perform simulation of T using the data points D_{sim}
  - Let e be the output error
- **else**
  - Let e be the prediction error of T calculated using D_{learn}
- **end**
- Let t be the terminal node of T corresponding to largest sum of squared errors
- Return (e, t)

The Lolimot method builds the tree structure using an iterative procedure (Nelles, 2001). In each iteration, the size of the tree, i.e., the total number of nodes, is increased by one. This is performed by converting a terminal tree node into an inner split node with two immediate descendants.

Let a *candidate split* s be a triple consisting of a terminal node of the tree built so far, a attribute to split on, and a split cut-point

$$s = (t, x_s, v_s) \tag{3.2}$$

In each iteration, several candidate splits are considered. The set of all candidate splits

considered is created using the following procedure: (a) one terminal node is selected for further splitting from the existing intermediate tree; (b) all attributes are considered as split attributes. The method only evaluates one cut-point per split attribute, i.e., it considers only "half-splits", which divide the current partition into two equal halves.

Let the terminal node t define a partition:

$$P_t = \{\mathbf{x} \in R^p | (v_1^{(0)} \leq x_1 < v_1^{(1)}) \wedge (v_2^{(0)} \leq x_2 < v_2^{(1)}) \wedge ...(v_p^{(0)} \leq x_p < v_p^{(1)})\} \tag{3.3}$$

The boundaries of the partition ($v_j^{(0)}$ and $v_j^{(1)}$) that the terminal node t defines, are used to calculate the cut-points of the candidate splits. The set of candidate splits, considered for expanding the terminal node t, is:

$$S = \{(t, x_1, \frac{v_1^{(0)} + v_1^{(1)}}{2}), (t, x_2, \frac{v_2^{(0)} + v_2^{(1)}}{2}), ...(t, x_p, \frac{v_p^{(0)} + v_p^{(1)}}{2})\}, \tag{3.4}$$

where $v_j^{(0)}$ and $v_j^{(1)}$ denote the boundaries of the partition in dimension j, as defined in Eq 3.3. Each of the candidate splits is then evaluated by using a heuristic greedy evaluation function consisting of three steps:

- A tree with the candidate split is created.
- Local models are estimated for the two new terminal nodes.
- The fit of the whole model tree to the training data is calculated.

For a tree of m terminal nodes, this procedure considers and evaluates only a small subset of all possible trees with $m + 1$ terminal nodes. One step of the iterative procedure is depicted in Figure 3.2.

Node and split selection heuristic. The Lolimot algorithm considers only one existing terminal node for expansion at a time. It selects the terminal node which gave rise to the largest sum of squared errors in the previous iteration. In each iteration, the model built so far is evaluated either using simulation or prediction, cf. the procedure **Evaluate** in the pseudocode shown in Algorithm 3.3. This evaluation is performed using all available training data (denoted as D_{sim}), and no averaging is performed on the individual squared errors. A heuristic of this kind favors terminal nodes which contain more training data over those which contain less.

The algorithm can choose among two different heuristics for selecting splits. The first is based on the selection heuristic component of the Lolimot algorithm, which is tailored for dynamic systems and uses simulation. It performs a simulation of the model in each iteration, using D_{sim}. In more detail, the algorithm uses the error of the simulation procedure for a) selection of the terminal node to further split in the next iteration, and b) selection of one among several candidate splits. The second alternative utilizes the one-step-ahead prediction performance of the model tree with the added candidate split.

3.4.1 Estimation of Local Models' Parameters

In each iteration the parameters of the newly added terminal nodes are estimated. The estimation begins by calculating the fuzzy membership function values. As a next step, these values are used in the weighted least square regression performed to obtain the parameters of the local models.

Fuzzy membership function. The parameters of the partitions defined by the model tree, i.e., the borders of the partitions, are used to define the fuzzy membership functions. The membership functions determine the (fuzzy) membership of each data point to each

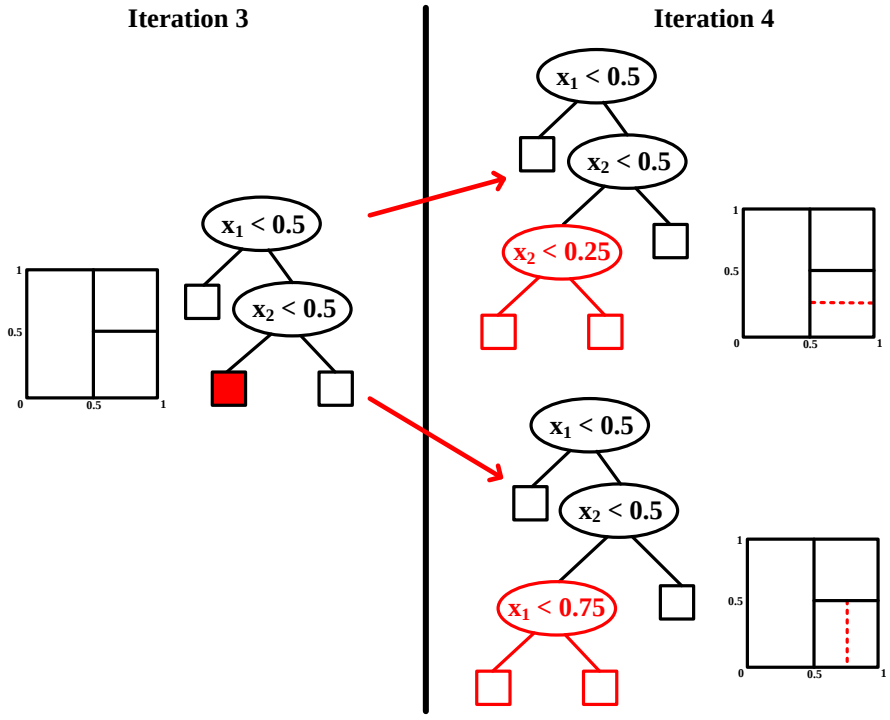


Figure 3.2: One iteration of the Lolimot method. The left-hand side depicts the model tree and its partitioning in the third iteration. The marked terminal node is the terminal node selected for further splitting. The right-hand side depicts the complete set of candidate splits considered in the fourth iteration and the corresponding partitionings of the input space.

of the partitions and the corresponding local models. The Lolimot method uses the multi-dimensional Gaussian membership function (Nelles, 2001), whose center $\mathbf{c}$ is set at the center of the partition, and standard deviation vector $\boldsymbol{\sigma}$ is calculated as 1/3 of the size of the partition (Nelles, 1999). For example, consider the tree of iteration 3, shown in Figure 3.2. Also, consider the partition corresponding to its leftmost terminal node, defined by $x_1 < 0.5$. The center of this partition, $\mathbf{c} = [0.25, 0.5]^T$, is the center of the corresponding membership function, while the standard deviation of the membership function is $\boldsymbol{\sigma}$= $[0.5/3, 1/3]^T$.

The membership of a data point $\mathbf{x}$ to the j-th partition is calculated as

$$\mu_j(\mathbf{x}) = exp(-\frac{1}{2}\sum_{i=1}^{n}(\frac{x_i - c_i}{\sigma_i})^2). \tag{3.5}$$

After the membership values $\mu_j(\mathbf{x})$ for a data point to all the partitions are calculated, these values are normalized across all partitions

$$\Phi_j(\mathbf{x}) = \frac{\mu_j(\mathbf{x})}{\sum_{k=1}^{m}\mu_k(\mathbf{x})} \tag{3.6}$$

thus obtaining the validity function values $\Phi_j(\mathbf{x})$.

Note that the j-th local model for the l-th target variable is determined by the vector of coefficients $\begin{bmatrix} b_{l,j,0} & b_{l,j,1} & \cdots & b_{l,j,p} \end{bmatrix}$, where p is the number of attributes. The parameter estimation determines the coefficients $b_{l,j,u}$ of the local models that correspond to the terminal nodes of the tree. In the multi-target case, this problem can be formulated as an optimization problem, with an objective function:

$$I = \sum_{l=1}^{r}\sum_{i=1}^{n} e_{i,l}^2 \tag{3.7}$$

where $e^2 = (y - \hat{y})^2$, r is the number of target variables and n is the number of data points.

Since the parameters of the local models for each target are independent of each other, we formulate r optimization subproblems:

$$I_l = \sum_{i=1}^{n} e_{i,l}^2 \quad l = 1, 2, .., r. \tag{3.8}$$

The parameter vector for l-th target variable contains $m \cdot (p + 1)$ parameters

$$\mathbf{b}_l = \begin{bmatrix} b_{l,1,0} & b_{l,1,1} & \cdots & b_{l,1,p} & \cdots & b_{l,m,0} & b_{l,m,1} \cdots & b_{l,m,p} \end{bmatrix}^T. \tag{3.9}$$

The total number of parameters that need to be identified for one model tree is $m \cdot r \cdot (p+1)$.

In the fuzzy modeling literature, two approaches are commonly used for the estimation of the local model parameters, given that the validity functions are known. The *local estimation* procedure (Johansen & Babuška, 2003) estimates the parameters of each local model in isolation of each other. An alternative is the *global estimation* procedure (Jang et al., 1997), which estimates all of the local model parameters simultaneously. The former one is faster but does not take the interactions between local models, as defined by the membership function, into account.

As previously discussed, the parameter estimations for each of the targets are treated as separate least-squares estimation problems. So, in the case of fuzzy local estimation, a fuzzy Lolimot model tree with m local models predicting r targets requires a total of $r \cdot m$ separate weighted least square regression problems to be solved.

For the l-th target and j-th local model, the regression matrix $\mathbf{X}_{l,j}$ is a $n \times (p+1)$ matrix, containing the values of the regressor variables. Its form is:

$$\mathbf{X}_{l,j} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,p} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,p} \end{bmatrix}. \tag{3.10}$$

The weighting matrix $Q_{l,j}$, composed of the validity function values (Eq. (3.6)) is:

$$Q_{l,j} = \begin{bmatrix} \Phi_j(\mathbf{x}_1) & 0 & \cdots & 0 \\ 0 & \Phi_j(\mathbf{x}_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Phi_j(\mathbf{x}_n) \end{bmatrix}. \tag{3.11}$$

Also, we define the vector of values for the l-th target variable, $\mathbf{y}_l$ as

$$\mathbf{y}_l = \begin{bmatrix} y_{l,1} & y_{l,2} & \cdots & y_{l,n} \end{bmatrix}^T, \tag{3.12}$$

where $y_{l,i}$ denotes the value of the l-th target for the i-th data point. Finally, the parameter estimates for the l-th target variable of the j-th local model can be calculated using the well-known weighted least squares estimation formula

$$\hat{\mathbf{b}}_{l,j} = (\mathbf{X}_{l,j}^T Q_{l,j} \mathbf{X}_{l,j})^{-1} \mathbf{X}_{l,j}^T Q_{l,j} \mathbf{y}_l. \tag{3.13}$$

3.4.2 Multi-target Lolimot Model Trees

This part summarizes the modifications of the Lolimot model tree learning algorithm necessary for handling multiple targets. Recall that the Lolimot algorithm was introduced in the system identification domain, and its multi-target version is used for modeling static and dynamic multi-output systems. The algorithm contains three modifications for multi-target modeling, as compared to the single-target version. These are:

- Determination of the set of candidate splits to evaluate. The algorithm chooses only one terminal node and evaluates half splits in all possible dimensions. The sum of squares is used as the criterion to choose the terminal node. While in the single-target case this is trivial, in the multi-target case, the squared errors for each target are normalized and summed. The terminal node which results in the largest error, i.e., sum of normalized squared errors, is selected. The normalization is performed for each node separately.

- Estimation of local model parameters. The algorithm evaluates a candidate split by building local models for the two new terminal nodes. The estimation of local model parameters is performed separately for each target variable, i.e., we have a separate optimization problem for each target (cf. Eq. (3.13)).

- Comparing performance of candidate splits in the multi-target case. After the local model parameters are estimated for each of the targets, and the model predictions are calculated for every target, the errors for each of the targets are aggregated. An aggregate multi-target root relative mean-squared error (RRMSE), also denoted as normalized root mean squared error (Nelles, 2001), is calculated as:

$$RRMSE = \sqrt{\frac{\sum_{l=1}^{r} \sum_{i=1}^{n} (\hat{y}_{i,l} - y_{i,l})^2}{\sum_{l=1}^{r} \sum_{i=1}^{n} (\bar{y}_l - y_{i,l})^2}} \tag{3.14}$$

where $\bar{y}_l$ is the mean value of the l-th target variable, while $\hat{y}_{i,l}$ and $y_{i,l}$ are the predicted and actual values respectively, for the l-th target variable and i-th data point.

3.4.3 Optimal Complexity of a Lolimot Tree

The optimal size of the single-target or multi-target Lolimot tree, in terms of number of local models, is determined by using the Akaike Information Criterion (AIC). The AIC criterion selects the algorithm iteration which provides the best trade-off between model complexity and accuracy. It takes into consideration the size of the training set n, the error of the model for each iteration and the total number $m \cdot p$ of coefficients in the linear models. The AIC calculated for a Lolimot model tree (Nelles, 2001) is:

$$AIC = n \cdot log(e_{MSE}) + m \cdot (p + 1) \tag{3.15}$$

where e_{MSE} denotes the mean-squared error of the model tree.

3.5 Properties of Existing Approaches

This section outlines the advantages and disadvantages of the existing tree-based approaches when used for dynamic system identification. It starts with the properties of model tree learning algorithms, and discusses the advantages and shortcomings of M5', Lolimot and other related tree learning approaches. After that, it discusses the properties of existing tree ensemble approaches.

3.5.1 Existing Tree Approaches

The existing tree approaches introduced earlier in this chapter were Lolimot and M5'. This subsection first discusses their properties, after which it presents other related tree approaches.

The advantages of the Lolimot method, which builds fuzzy model trees, are summarized as:

- The complete model is evaluated, so the interactions between the local models in a fuzzy setting are taken into account.
- The output error is evaluated during split selection, instead of the prediction error, which might produce a model with better simulation performance.
- The local parameter estimation produces models with good noise handling capabilities (Nelles, 1999).

On the other hand, the potential limitations of Lolimot are related to the complexity of the produced model: The required number of terminal nodes to obtain a model with a desired accuracy may not be optimal. They can be summarized as:

- Only half splits are considered, i.e., the split cut-point is not optimized, and the MSF overlap is not optimized. This in turn might produce fuzzy model trees with more local models, as compared to approaches which optimize the split cut-point and MSF overlap.

Several existing methods build upon Lolimot and consider replacing the type of splits with axes-oblique, and the heuristic split selection with a genetic programming approach. The first extension (Nelles, 2006) includes a modification which considers axes-oblique

splits, and not only axes-parallel, as in this thesis. It uses a more computationally expensive nonlinear optimization technique to determine the optimal position and direction of each split. However, on a simulated example, the differences to a standard Lolimot model are visible: the size of the tree is almost a third of the size of the standard Lolimot model, for a pre-determined model error. The consideration of axes-oblique splits thus presents a tradeoff between the faster training time of Lolimot and the smaller size of the oblique model trees.

The second extension (Hoffmann & Nelles, 2001) introduces Genetic Programming (GP) for determination of the structure of the Lolimot model tree. Also, it considers different cut-point for the splits, as compared to the single cut-point, i.e., half-split. The conclusions are that the GP method may again reduce the size of the model tree, while obtaining comparable predictive performance to a standard Lolimot model. However, the number of possible partitionings that the method has to evaluate is very large, so its applicability is limited due to the large time required for learning.

The model tree learning algorithm M5' has different properties. It builds crisp trees, is fast and efficient. Its advantages can be summarized as:

- The split selection procedure is fast and linear in the number of features.
- The local model estimation is fast, since the number of data points used is smaller than in the fuzzy case.
- It can handle discrete attributes and missing values in the attributes.

However, the accuracy of M5' model tree is in many cases lower, as compared to Lolimot. This might be due to the lack of look-ahead step during the split selection, or the crisp and not fuzzy local model estimation. The limitations of the M5' algorithm can be summarized as:

- The model trees typically have discontinuities on the borders of local models.
- The split selection procedure does not include look-ahead.
- The split selection procedure may show potentially pathological behavior for low-dimensional datasets (Vens & Blockeel, 2006).

Other tree learning approaches. After describing the advantages and disadvantages of M5' and Lolimot, this part will outline the properties of other related tree learning approaches. The *spatial piecewise linear models* of Billings and Voon (1987) are a nonlinear identification technique, which use interpolation of local linear models. The model representation is equivalent to the crisp model trees, discussed above, since there is no overlap of the operating regions. The operating regions are defined by using a grid partitioning, and the parameters of the linear models are estimated using least squares.

A clear disadvantage is that the only type of partitioning considered is grid partitioning, which may potentially lead to a large number of regions. The authors also conclude that some of the potentially large numbers of operating regions may not be reachable (no training data for those regions can be obtained, as the system can not be in that state), hence they do not identify local models for all regions. However, they note that since the local models are independent of one another, due to the crisp splitting, linear stability criteria could be applied to evaluate the local models that are identified.

The spatial piecewise linear models also expect a predefined number of operating regions. This is determined by the user's choice to divide the input and output data into several equal intervals. This is in contrast to the automatic determination of the number

and position of operating regions found in some of the other approaches, like the model tree algorithms, discussed earlier, use.

The *regime decomposition* approach of Johansen and Foss (1995) uses a tree approach to search for the optimal structure and parameters. This approach is similar to the Lolimot algorithm, and the final model is also in the form of a TS model. The aim of the algorithm is to determine the optimal axis-orthogonal decomposition into operating regions.

However, there are several differences as compared to the Lolimot approach. The algorithm is more oriented towards optimizing the complexity (number of local models required), so it tries to optimize the cut-points for the splits, something that Lolimot does not perform. Another difference is the proposal of an extended horizon search strategy. This type of search is a deep look-ahead, where an optimal split is determined by looking at successive n^* splits (n^* is the search horizon), the partitioning they produce, and the predictive performance of the local models that are built. In comparison, the existing model tree algorithms discussed so far use only a shallow look-ahead or greedy search by only evaluating and building one split and the predictive performance of its immediate descendant local models.

3.5.2 Existing Ensembles of Model Trees and Their Limitations

The ensemble approaches of bagging (Breiman, 1996) and Random Forests (Breiman, 2001), which are used in this thesis, have been originally designed to use classification and regression trees. Since bagging only manipulates the training data, and does not randomize the base learners, it can be used with any base learner, including algorithms that learn model trees. However, only a few authors have considered learning ensembles of model trees, instead of regression trees. To our knowledge, at least two ensemble approaches using crisp model trees have been introduced already.

The first is the semi-random model tree ensemble approach of Pfahringer (2011). This approach modifies the base-level tree-learning algorithm to produce balanced trees: the number of points falling in each terminal node of the tree is approximately the same. This approach is thus not well-suited for dynamic systems, whose identification is performed on data that are not evenly distributed in the instance space. The partitioning of the instance space using semi-random model tree ensembles would be denser around the equilibrium points, as these regions contain more points than the out-of-equilibria regions. As a consequence, the critical out-of-equilibria regions would be covered by a small number of partitions, resulting in poor approximations.

The second approach is the model tree ensemble method of Jung, Reichstein, and Bondeau (2009). It introduces ensembles of crisp model trees, by learning trees with random splits. The model trees are built using a new model tree induction algorithm, named TRIAL. The base learning algorithm uses randomization, i.e., its operation is perturbed in order to increase the diversity of the predictions.

Chapter 4

Model Trees and Ensembles for Dynamic System Modeling

This chapter introduces modifications and extensions to existing model tree algorithms. These are aimed at improving the performance of model trees for modeling dynamic systems. The modifications to the Lolimot model tree algorithm address the split selection, as well as the computational efficiency of the method. Also, ensembles of Lolimot model trees are presented, which are now applicable, given that the modified base learning algorithm is faster.

The modifications to Lolimot, are inspired by the *generalization power of the soft model trees as opposed to the crisp ones*. The modifications introduced are summarized as an approach named L++, which evaluates more candidate splits, and also improves the computational efficiency over the existing Lolimot method.

The aim of the modifications was to reduce the size of Lolimot trees, i.e., the number of local models, as well as to improve the efficiency over Lolimot, while retaining similar predictive power. In other words, the modifications proposed to the model tree learning algorithm modify the methodology, since a different search heuristic is employed, and modify the implementation, since the split evaluation is replaced with a more efficient variant. Also, by using the more efficient L++ method, one is also able to build ensembles, as the learning times of the base algorithm are reduced.

The aim of this work is to introduce modifications of model tree algorithms for dynamic systems, and this in turn requires regression approaches that produce close fits to smooth static functions (cf. Subsection 2.1.1.2, which discusses the external dynamics approach). For this, the applicability of crisp model tree approaches is limited, as they are not well suited for approximating smooth nonlinear functions. However, this work would also try to show that to a certain extent, two modifications of crisp model trees could be used for modeling dynamic systems: a) post-smoothing, and b) ensembles, which correct the local model estimates of single crisp trees. After the empirical evaluation, which is presented in the following sections, this work will try to draw some conclusions regarding their utilization.

This chapter is organized as follows. It starts with the modifications to the crisp model tree approach M5', for improving its fit to smooth nonlinear functions. Then it discusses the modifications to a soft model tree approach, Lolimot. Next, it presents the ensembles of model trees, based on the bagging principle. Finally, it illustrates the difference in predictive power of fuzzy over crisp model tree approaches, as well as model tree ensembles.

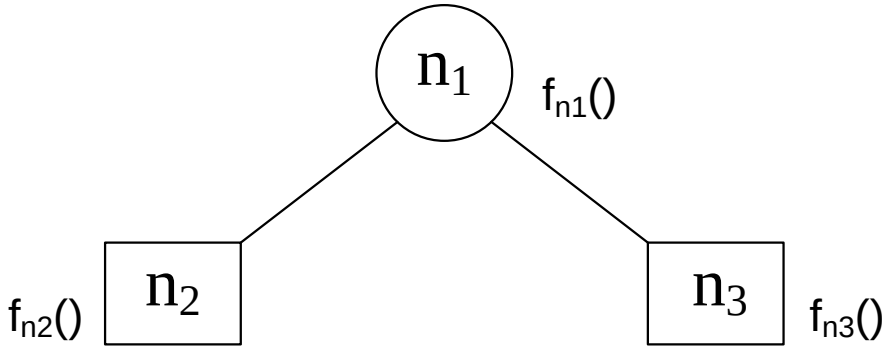


Figure 4.1: An example model tree with one split node and two terminal nodes.

4.1 Crisp Model Trees

An example of a crisp model tree is shown in Figure 4.1. It consists of one split node and two terminal nodes. During the learning with the model tree algorithm M5', local linear models are learned for all three nodes. The final prediction of the crisp model tree is calculated either using node $n2$ or node $n3$, i.e., the prediction for the feature vector $\mathbf{x}$ is either $\hat{f_{n2}}(\mathbf{x})$ or $\hat{f_{n3}}(\mathbf{x})$. In more general terms, a crisp model tree consisting of m terminal nodes and $m-1$ splits, which form m partitions $P_1, P_2, \cdots, P_m$, is a piecewise linear model of the form:

$$f(\mathbf{x}) = \begin{cases} f_{LM_1}(\mathbf{x}) & \text{if } \mathbf{x} \; is \; in \; P_1 \\ f_{LM_2}(\mathbf{x}) & \text{if } \mathbf{x} \; is \; in \; P_2 \\ \vdots & \\ f_{LM_m}(\mathbf{x}) & \text{if } \mathbf{x} \; is \; in \; P_m \end{cases} \tag{4.1}$$

In the following, we will consider two different techniques for smoothing the predictions of a crisp model tree, built by the M5' algorithm.

4.1.1 Smoothing the Crisp Model Tree Predictions

The smoothing of the model tree predictions can be done by using different strategies. This work considers two types, both of which are executed after the crisp M5' model tree is built and its local models are estimated. The first is the built-in M5' smoothing, which is only a correction of the local model coefficients. The other type is an interpolation approach, which performs smoothing using fuzzy weighting, and changes the form of the model from a crisp to a soft model tree.

The first variant corrects the local model coefficients for a terminal node of the tree, by using the local models found on the path from that node to the root. The resulting model stays a crisp model tree, i.e., only one local model is used to obtain the model tree prediction (cf. Eq. (4.1)). The second variant performs the smoothing by using a weighted sum of the local model predictions of all terminal nodes of the tree. This means that a soft model tree, in this case, uses all local models to obtain the model tree prediction (cf. Eq. (4.4)).

The smoothing could potentially increase the accuracy of the model tree, when fitting smooth functions, and could also address the boundary discontinuities issue. The latter issue appears on the boundaries between adjacent local models, which are potential places

where discontinuities may appear. Two neighboring local models may have different predictions for "close" input vectors, from the other side of the boundary. For example, a discontinuity appears for the value of $x = 0.5$ in the left part of Figure 4.8. In the rest of this subsection, a detailed description of the two smoothing approaches follows.

4.1.1.1 The Built-in M5' Smoothing

The *built-in M5' smoothing* is a procedure aimed to improve the accuracy of model tree predictions (Quinlan, 1992). Its goal is to smooth the response of local models built in leaf nodes with a small number of data points, by also considering the predictions of the local models built in other internal tree nodes.

It is an iterative bottom-up procedure, performed after the initial tree has been built and pruned. The procedure starts by correcting the local model corresponding to a terminal node of the tree, by using the linear models of all tree nodes on the path up to the root. Assume that the local model for the terminal node $n2$ is $\hat{f}_{n2}()$, and the number of data points sorted down to that node is d. Then, the smoothed prediction, $\hat{f}'_{n2}()$ would be calculated using the local model of node $n1$, as:

$$\hat{f}'_{n2}(\mathbf{x}) = \frac{d\hat{f}_{n2}(\mathbf{x}) + k\hat{f}_{n1}(\mathbf{x})}{d+k} \tag{4.2}$$

where k is some constant value. If we rearrange, and introduce $\alpha = \frac{d}{d+k}$, we obtain:

$$\hat{f}'_{n2}(\mathbf{x}) = \alpha\hat{f}_{n2}(\mathbf{x}) + (1-\alpha)\hat{f}_{n1}(\mathbf{x}). \tag{4.3}$$

This procedure only performs a "correction" of the possibly inaccurate local model predictions of terminal nodes with a small number of data points. We can observe that the weighting factor α is small in case of a leaf node with small number of data points d, and is big otherwise. This technique is implemented in M5', as only a correction to the coefficients of the local models in the terminal nodes of the tree. This means that the model tree does not change its form, i.e., it remains a crisp model tree. Additionally, the value of the constant k is set to 15 in the WEKA implementation of the M5' algorithm. It is worth noting that this approach only partially solves the discontinuity problem, as it only compensates for the large jumps of the predictions, and does not provide a continuous transition on the boundaries between local models.

4.1.1.2 Smoothing Using Fuzzification

The smoothing of the linear models in a crisp model tree can be achieved by an alternative approach, using fuzzification, and converting the model to a soft model tree. The original crisp model tree (4.1) is converted to a soft model tree which has the form:

$$f(\mathbf{x}) = \sum_i w_i(\mathbf{x}) f_{LMi}(\mathbf{x}) \tag{4.4}$$

where $\sum w_i(\mathbf{x}) = 1$. The difference to the M5' built-in smoothing, described earlier, is that in Eq. (4.3), α is not a function of the feature vector $\mathbf{x}$, while here, w is a function of $\mathbf{x}$. This means that the fuzzy model tree presents a different formalism than the one described above.

The *smoothing using fuzzification*, which was introduced by Jang (1994), and later applied to M5' model trees (Aleksovski, Kocijan, & Džeroski, 2013, 2014c), is performed by representing the splits $s[x_j < c]$ of the crisp model tree with a fuzzy set characterized

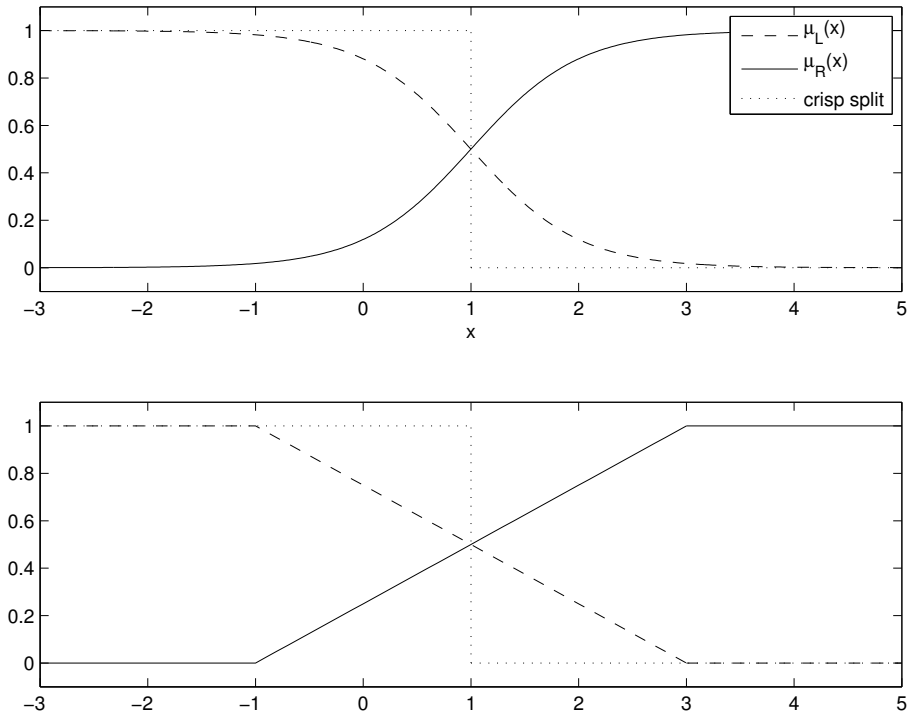


Figure 4.2: Two types of fuzzy membership functions: sigmoidal (top) and triangular (bottom).

by some membership function (MSF). For example, a crisp split of the form $s[x_j < c]$, where x_j is the j-th feature, and c is a cut-point, is transformed to a fuzzy split by using a sigmoidal membership function $\mu(x_j, c, \alpha)$ with a fuzzy parameter α (the inverse split width) :

$$\mu_L(x_j, c, \alpha) = \frac{1}{1 + \exp(-\alpha(x_j - c))} \tag{4.5}$$

$$\mu_R(x_j, c, \alpha) = 1 - \mu_L(x_j, c, \alpha) \tag{4.6}$$

Several types of fuzzy membership functions can be used (triangular, sigmoidal, Gaussian), two of which are shown in Figure 4.2. The choice between any of the three membership functions should provide similar prediction results. In the case when the parameters of each MSF are optimized globally (the parameters of all MSFs are optimized in a single optimization procedure), differences in performance may appear. However, this work calculates the MSF parameters by using a predefined percentage of the partition size.

In other words, for the sigmoidal membership function of Eq. (4.5), the value of α is calculated such that the overlap between the two subpartitions is equal to a predetermined percentage $p_{overlap}$ of the size of the partition in the dimension of the split attribute. Larger values for $p_{overlap}$ mean smoother transitions between the local models. The optimal value of $p_{overlap}$ is different for each modeling problem, i.e., needs tuning.

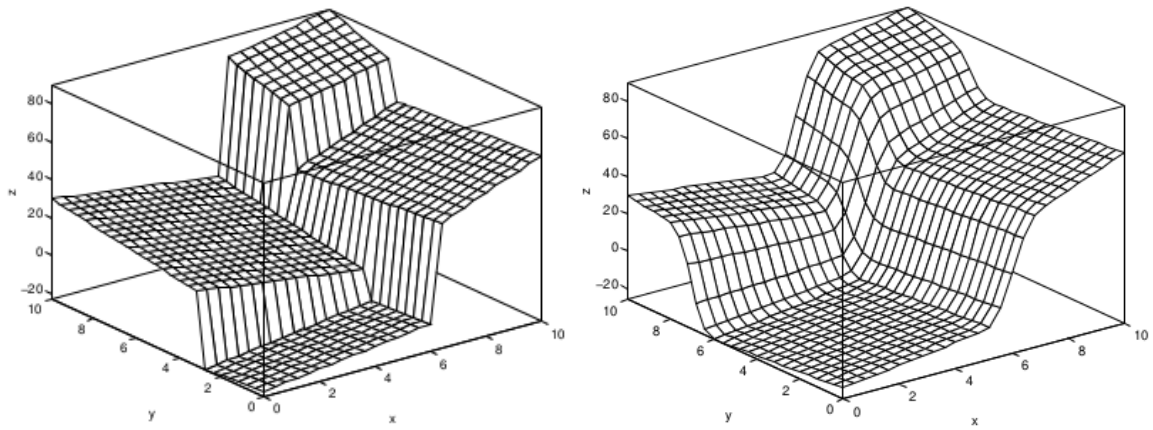
The prediction of the model tree of Figure 4.1 with one fuzzy split $\mu(x_j, c, \alpha)$ and two local models f_{n2} and f_{n3} is calculated by using the following formula:

$$\hat{f}(\mathbf{x}) = \mu(x_j, c, \alpha)\hat{f}_{n2}(\mathbf{x}) + (1 - \mu(x_j, c, \alpha))\hat{f}_{n3}(\mathbf{x}) \tag{4.7}$$

In general, a data point $\mathbf{x}$ is associated with all LMs (terminal nodes) of a soft model tree, but with different weights. The weight for a LM is calculated by multiplying the

μ membership function values for all of the splits from the root of the tree down to the particular terminal node.

The benefits of smoothing a crisp model tree with fuzzification are the potentially more accurate predictions, as compared to the crisp model tree. This is illustrated in Figure 4.3, where the resulting models of crisp and smoothed model trees with 4 LMs are depicted.



(a) Crisp model tree. (b) Smoothed model tree.

Figure 4.3: Comparison of crisp and smoothed model trees with 4 LMs, in a 2-dimensional space.

4.1.2 Multi-target M5'

This part describes the modifications of M5' for predicting multiple dependent variables, i.e., targets. It presents the modifications performed to the split selection heuristic in the tree growing phase, and the error calculations in the tree pruning phase.

The standard deviation reduction (SDR) is calculated by aggregating the SDR of all target variables. For a given candidate split attribute, the SDR heuristic is first calculated for each target, using Eq. (3.1). The aggregated heuristic value SDR_{MT} is then calculated as:

$$SDR_{MT} = \frac{1}{r} \sum_{j=1}^{r} SDR'(j) \tag{4.8}$$

where $SDR'(j)$ is the SDR value for the j-th target, scaled to the interval $[0, 1]$. The candidate split with the largest SDR_{MT} is selected and used by the M5' algorithm.

The pruning phase of M5' is also modified for the multi-target scenario. The calculation of the linear models is performed independently for each of the targets. The overall error of the subtree, is calculated as an average of the subtree errors of each target. The overall error of the linear models for the r targets is calculated as an average of the errors of each target. The error values are normalized before both averaging operations.

4.2 Fuzzy Model Trees

In this part, we are going to consider fuzzy model trees, as built by the Lolimot method. Lolimot was introduced in the previous chapter, and here we are going to introduce the modifications performed to Lolimot. Two of these concern the structure search: a more

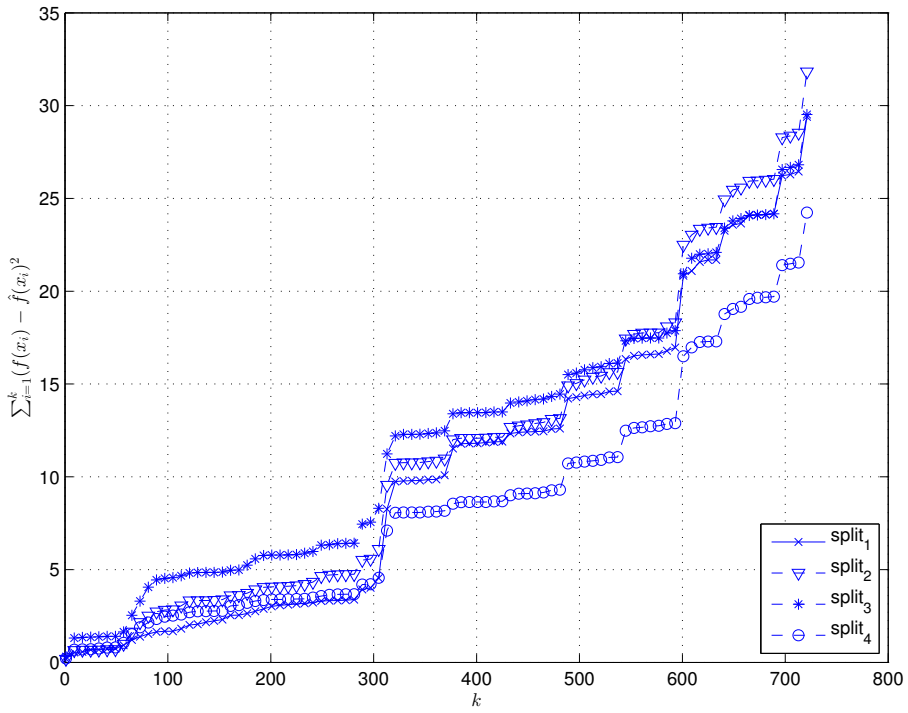


Figure 4.4: Evaluating four candidate splits using simulation, during building the Lolimot tree for GLS. The x axis denotes the discrete step k of the simulation procedure, while the y axis the running sum of squared errors $\sum_{i=1}^{k}(f(x_i) - \hat{f}(x_i))^2$.

efficient evaluation of candidate splits and a modified search for the optimal tree structure. The latter is performed by considering several cut-points when generating the candidate splits and using different values for the fuzzy overlap. The other two modifications concern the split evaluation and the estimation of local models.

4.2.1 Modifying the Evaluation of Candidate Splits

This modification considers the evaluation the candidate splits in Lolimot. The evaluation of each of the candidate splits is used to select the split which results in a smallest estimated error and added this split to the tree. The candidate splits in Lolimot are evaluated by creating the intermediate tree with the added split, estimating the two new local models, and performing simulation on the training data. The procedure in Lolimot is executed on the complete set of training data points, and results with an estimate of the effectiveness of the candidate split.

We propose to make the estimate of effectiveness of candidate splits faster, by modifying the evaluation process. The modified evaluation would only consider a part of the training data, and not the full training set. In the following we describe the details of our proposed modification, and its advantages.

Instead of executing the simulation procedure using all training data points, it is our experience that differences between the performance of intermediate trees (corresponding to the candidate splits) are visible much earlier. For example, Figure 4.4 displays the simulation procedure for four intermediate trees, when building a model for the pressure variable of the GLS system. The y axis of the figure displays the running sums of squared error, for the four different intermediate trees, while the x axis the number of steps in the simulation procedure.

It can be seen that the optimal intermediate tree, calculated by simulating on the full training set, corresponds to candidate split 4. This selection result can also be determined by only performing the simulation half-way, i.e., only to step $k_{SIM} = 350$. For this example, the simulation results for the subsequent data points beyond this step do not change the outcome of the selection. So, one can conclude that early stopping in the simulation procedure can be used as a modified estimate, for ranking the candidate splits. The advantage of using the modified heuristic is that the running time of the evaluation part is reduced. The disadvantage is that it might not result in the optimal split, as seen by Lolimot's current estimate, for splits with similar performance. This might happen if k_{SIM} is too small, or the splits show very similar performance.

The fact that the differences in performance are visible earlier in the simulation procedure might be explained with the error accumulation phenomenon. The error accumulation increases the error of the worse-performing intermediate trees much earlier in the simulation procedure (penalization). This might allows an easier determination of the better performing split or splits in the selection procedure.

The reason that the efficiency of the split evaluation heuristic is increased by using early stopping in the simulation procedure, is that the evaluation of a soft model tree is computationally expensive. This stems from the fact that for one data point the Lolimot model tree needs to calculate all membership function values, as well as the predictions of all local models. In general, setting k_{SIM} to smaller or lager values may be considered as a tradeoff between speed of learning and accuracy of the obtained model.

4.2.1.1 Utilization of the Output Error While Learning

Related to the evaluation of the candidate splits, we also modify the method so that the evaluation of candidate splits could be performed by either the output or the prediction error. The former is the default when evaluating splits for dynamic systems, in the Lolimot method. By implementing the prediction error evaluation, we hope to get more insight in the differences between simulation and one-step-ahead prediction, when used during the model building procedure. Please note that, in either case, the estimation of the local model parameters is performed by minimizing the prediction error.

4.2.2 Modifying the Search for an Optimal Tree Structure

When comparing the model tree approaches introduced in the machine learning domain to Lolimot, one difference that can be noticed is that the Lolimot method evaluates only a small number of candidate splits. For each feature variable (dimension) it evaluates only one half-split. A structure search for the optimal tree structure, which evaluates more different candidate splits, has the potential to discover smaller model trees with acceptable performance.

However, in the context of soft or fuzzy trees, and TS models, the model structure is determined by the number, the position and the overlap of the fuzzy membership functions. This is why this part is going to introduce modifications which evaluate a) several different split cut-points, and with this, several different positions of the fuzzy MSFs, and b) different overlap of the MSFs.

4.2.2.1 Considering Several Split Cut-points

Recall that in Table 3.2 we categorized the different implementations of the candidate cut-points determination, for a given feature attribute x_i. The conclusion there was that the

methods introduced in the system identification domain do not base the candidate cut-points determination on the data sample. Instead, they have an interval-oriented approach, and generate uniformly distributed candidate cut-points in the $[x_i^{min}, x_i^{max}]$ interval.

The modification introduced here is also interval-oriented, as it introduces q candidate cut-points for each variable. Values for q of up to 8 are used and evaluated. For example, given a terminal node t a feature attribute x_i, i.e., a data dimension i, the candidate splits considered are:

$$S_{t,i} = \{(t, x_i, x_i^{min} + \frac{1}{q+1}s), (t, x_i, x_i^{min} + \frac{2}{q+1}s), \cdots, (t, x_i, x_i^{min} + \frac{q}{q+1}s)\}, \quad (4.9)$$

where $s = x_i^{max} - x_i^{min}$ is the size in dimension i.

4.2.2.2 Considering Different Overlaps

The value of the vector $\sigma = [\sigma_1, \sigma_2, \cdots, \sigma_p]$ in Lolimot defines the deviation of the fuzzy membership function, and thus the amount of overlap for the local model in question. It is calculated as a fraction of the size of the partition in the corresponding dimension:

$$\sigma = k_\sigma [\delta_1, \delta_2, \cdots, \delta_p]^T \quad (4.10)$$

where δ_j is the size of the partition in the dimension j. This means that the algorithm does not optimize the overlap value for each of the LMs, but instead uses an overlap parameter, k_σ for this purpose. Other methods which build TS models consider different values for the overlaps of each MSF, and with this, optimize more parameters, as compared to Lolimot. An example is the well-known ANFIS method (Jang et al., 1997), which optimizes $2p$ parameters for each MSF, i.e., a total of $2pm$ MSF parameters for a TS model with m LMs.

In general, the fuzzy modeling literature also notes that setting overlap parameter to a large or small overlap between local models may have an impact on the parameters estimated for the local models. This means that this parameter could influence the number of LMs needed to accurately approximate some function. In other words, the overlap parameter could have an influence on the size of the final Lolimot tree.

As discussed, the Lolimot algorithm used the multi-dimensional Gaussian membership function. The standard deviation vector σ of this function is calculated by using the overlap parameter $k_\sigma = 1/3$ of the the size of the partition, in each dimension. Note, also that the amount of overlap, i.e., σ, is directly influenced by the partition boundaries, i.e., the split thresholds, as shown in Figure 4.5. Here, the values of $\sigma = [\sigma_1, \sigma_2]$ are directly dependent on the partition size in both dimensions.

In the analysis, we consider the effect of using different values for k_σ. We utilize values for k_σ ranging from 0.25 to 4. Additionally, we investigate what is the effect of tuning this parameter for different datasets.

4.2.3 Global Parameter Estimation in Lolimot

As discussed earlier, the estimation of the local model parameters of a model tree, given that the validity functions are known, could be performed using local or global parameter estimation. The *global estimation* procedure estimates all of the local model parameters simultaneously. This takes into account the interaction between local models, as defined by the membership function overlaps. The alternative is *local estimation*, which estimates parameters of each local model in isolation, and is used in the Lolimot method.

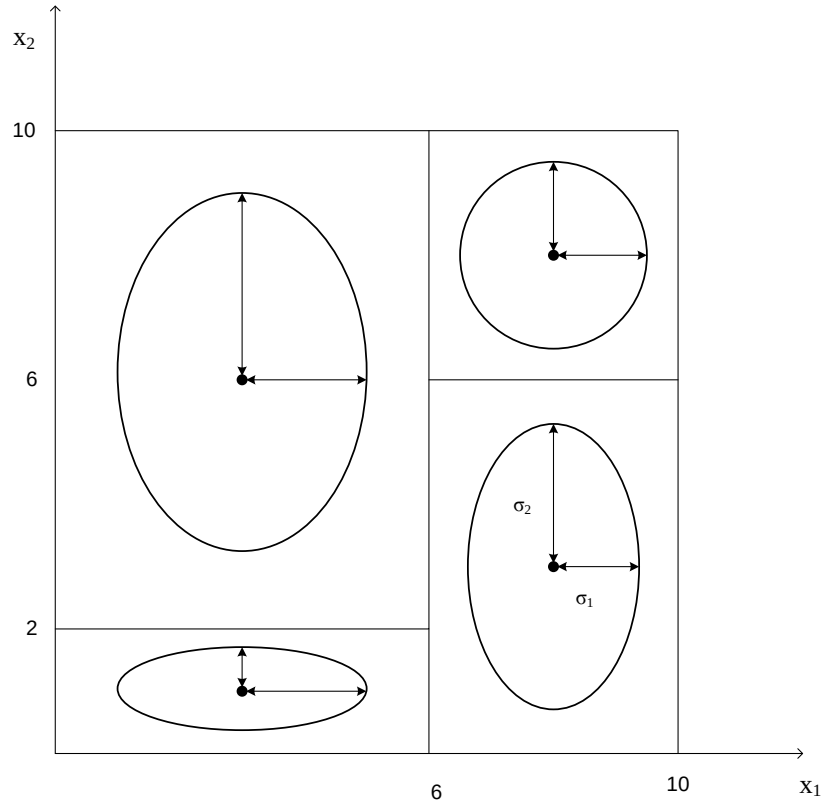


Figure 4.5: The partitioning resulting from a Lolimot model tree with three splits and four LMs. The Gaussian membership functions are shown too. For the bottom-right partition, with dimensions $[\delta_1, \delta_2] = [4, 6]$, the σ_1, and σ_2 values are shown.

The global estimation was already used in the Lolimot method, by Aleksovski, Kocijan, and Džeroski (2014b), where the resulting model tree had a small complexity and acceptable fit to the dynamic system. Additionally, the work of Nelles (1999) notes that the global estimation has two disadvantages. The first one is not handling noise well, and the second one is not providing for interpretation of the local model coefficients as local linearizations of the model. In more detail, the local model coefficients that this procedure estimates, have no physical interpretation for the underlying system.

The local estimation has faster calculation as an advantage, compared to the other alternative, but the interactions between local models are not taken into account. We aim to modify the implementation of the Lolimot algorithm, so that the LM estimation can also be performed with the global parameter estimation procedure. In the remainder of this part we outline the calculations required to perform the global parameter estimation for the multi-target case.

For the l-th target variable, the regression matrix $\mathbf{X}_l$ is composed of m submatrices, one for each local model:

$$\mathbf{X}_l = \begin{bmatrix} \mathbf{X}_{l,1}^{(\text{sub})} & \mathbf{X}_{l,2}^{(\text{sub})} & \cdots & \mathbf{X}_{l,m}^{(\text{sub})} \end{bmatrix}. \tag{4.11}$$

Given that $\Phi_i(\mathbf{x})$ denotes the validity function value, as defined in Eq. (3.6), each of the submatrices $\mathbf{X}_{l,i}^{(sub)}$, which consist of N rows and $(p+1)$ columns, have the following form:

$$\mathbf{X}_{l,i} = \begin{bmatrix} \Phi_i(\mathbf{x}_1) & \Phi_i(\mathbf{x}_1)x_1 & \cdots & \Phi_i(\mathbf{x}_1)x_p \\ \Phi_i(\mathbf{x}_2) & \Phi_i(\mathbf{x}_2)x_1 & \cdots & \Phi_i(\mathbf{x}_2)x_p \\ \vdots & \vdots & \ddots & \vdots \\ \Phi_i(\mathbf{x}_n) & \Phi_i(\mathbf{x}_n)x_1 & \cdots & \Phi_i(\mathbf{x}_n)x_p \end{bmatrix}. \tag{4.12}$$

We denote with $\mathbf{y}_l$ the vector of values of the l-th target variable, which is:

$$\mathbf{y}_l = \begin{bmatrix} y_{l,1} & y_{l,2} & \cdots & y_{l,n} \end{bmatrix}^T. \tag{4.13}$$

In Eq. (4.13), $y_{l,i}$ denotes the value of the l-th target variable for the i-th data point. The parameter estimates for each target can now be calculated using the well-known least squares estimation formula:

$$\hat{\mathbf{b}}_l = (\mathbf{X}_l^T \mathbf{X}_l)^{-1} \mathbf{X}_l^T \mathbf{y}_l. \tag{4.14}$$

4.3 Model Tree Ensembles

Ensembles for regression, also called committees of predictors, are known to improve predictive accuracy. This is known in the field of neural-network ensembles (Krogh & Vedelsby, 1995) as well as tree-based ensembles (Breiman, 1996). Among the reasons for their success are the smoothing effect on individual model estimates and the reduction of variance of the ensemble (Grandvalet, 2004), as compared to the one of the individual trees. The Model Tree Ensembles we are proposing here are based on the bagging (Breiman, 1996) principle, and use either ensembles of crisp M5' model trees, or ensembles of soft Lolimot model trees.

4.3.1 Ensemble Construction

The ensemble construction procedure we use is based on the popular bagging approach. As shown in Figure 4.6, bagging creates t bootstrap replicates, i.e., random samples with replacement, of the identification set, which have an equal number of data points as the

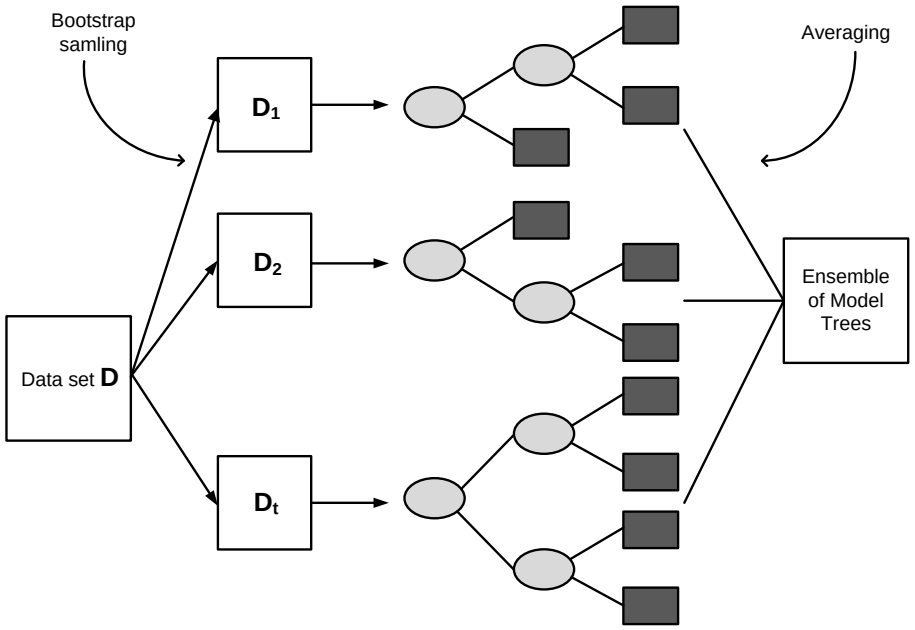


Figure 4.6: Operation of the bagging method, using model trees as base models.

identification set. Using each of the t samples, a base learner algorithm is used to build a collection of t model trees: $f_1, f_2, .., f_t$. Finally, the ensemble structure is optimized, using an ensemble selection procedure, as described in Subsection 4.3.2.

The final ensemble model is used for prediction by averaging the predictions of each of the base models. The pseudocode describing the ensemble construction procedure and ensemble selection procedure is shown in Algorithm 4.1. The base learner algorithm, denoted by the procedure **Build_tree**(D_i), can be either M5' or Lolimot. The operation of both of them has been described in the preceding text.

4.3.2 Ensemble Selection

After the ensemble is built, it is optimized by using a greedy ensemble selection procedure. Trees that do not contribute to the accuracy of the ensemble are removed from the ensemble. A tree's contribution is evaluated by considering the output error of the reduced ensemble without the tree and comparing its performance to the current ensemble. By evaluating the output error of the ensemble on the identification data (instead of the prediction error), we aim to produce a more successful model of the dynamic system. In the next paragraph we describe the ensemble selection procedure in more detail.

The selection procedure operates in a greedy fashion, reducing the ensemble size by one tree in each step, as shown in Algorithm 4.1. It stops when no improvement can be made to the performance of the ensemble. For dynamic systems, simulation on the identification data is performed and the evaluation of the performance of the ensemble is carried out by using the output error function.

After the ensemble selection procedure, assume that the resulting ensemble has t' trees: $E = \{T_1, T_2, ..T_{t'}\}$. The prediction of the ensemble is a uniformly weighted average of the model tree predictions:

$$\bar{f}(\mathbf{x}) = \frac{1}{t'} \sum_{i=1}^{t'} \hat{f}_i(\mathbf{x}) \tag{4.15}$$

where $\hat{f}_i(\mathbf{x})$ denotes the prediction of the i-th model tree for data point $\mathbf{x}$.

Algorithm 4.1: Pseudocode for the Model-Tree Ensembles method.

Algorithm Learn_ensemble(D)
  Data: data set D
  Result: an ensemble E
  Create t bootstrap samples of D : $D_1, D_2, .., D_t$
  Build a tree using each of the t samples: $T_i =$ **Build_tree**(D_i)
  Let $E' =$**Ensemble_selection**($\{T_1, T_2, .., T_t\}$, D)
  Return the ensemble E'
Algorithm Ensemble_selection(E, D)
  Data: ensemble E, consisting of trees $T_1, T_2, ..$, data set D
  Result: ensemble E'
  Let e_{full} be the output error of E, obtained by simulation on data set D
  Let $t = |E|$
  Create t ensembles, $E_1, .., E_t$ where $E_i = \{T_j | j \neq i\}$
  Let e_i be the output error of ensemble E_i
  Let $e_{reduced} = min_{i=1..t}(e_i)$
  Let $j = argmin_{i=1..t}(e_i)$
  if $(e_{full} > e_{reduced})$ **then**
    Return the ensemble **Ensemble_selection**(E_j, D)
  else
    Return the ensemble E
  end

4.4 Illustrative Example

This part is going to illustrate the predictive power of model trees and ensembles on a simple 1-dimensional static system. This will help understand the advantages and limitations of model tree algorithms for modeling dynamic systems. It will also illustrate the derivatives of the different types of models, as they are important for assessing the generalization performance.

The predictive power of the model trees ensembles is illustrated with the regression problem of fitting the static nonlinear function

$$f(x) = \sin(2\pi x) + 2x \tag{4.16}$$

using 200 points (x, y), with x uniformly distributed in the interval $[0, 1]$.

Figure 4.7 shows two fitting scenarios, where the models are learned using a noisy version of the data: noise with a standard deviation equal to 20% of the target variable deviation was added. In the first scenario, the nonlinear function is approximated with one Lolimot model tree with 12 LMs. The second scenario presents the fit of a bagging ensemble using randomized splits, where the base models are Lolimot trees. In both cases the fit is acceptable, while the error of the ensemble is slightly smaller. Both models handle the noisy data well, i.e., do not overfit to the noise.

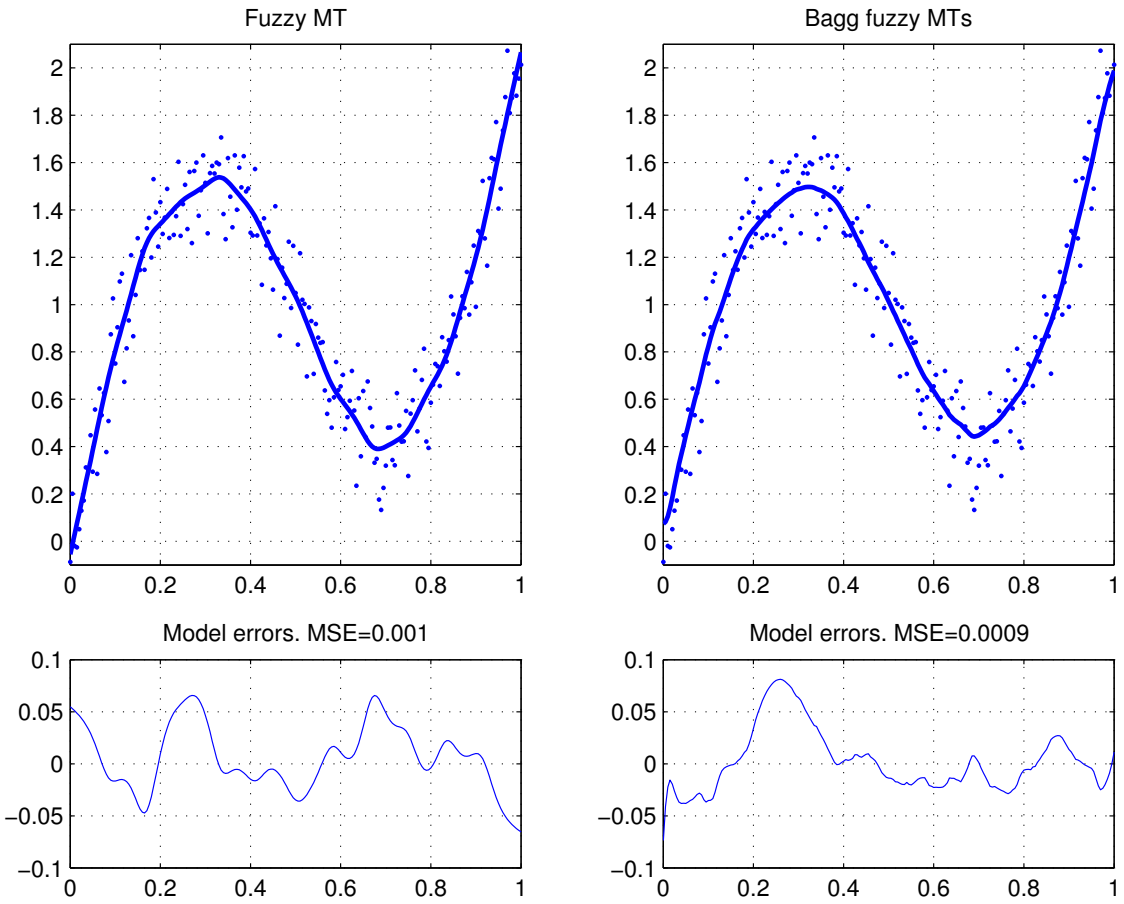


Figure 4.7: Performance of the soft model tree approaches. A single Lolimot tree (left) and bagging of Lolimot trees (right). The Lolimot model tree consists of 12 LMs. The bagging consists of 50 model trees with 12 LMs. The lower panels show the approximation error $f(x) - \hat{f}(x)$. In both cases the modeling is performed by using data with 20% noise.

Figure 4.8 shows the fit of a crisp M5' tree with 12 terminal nodes, the same crisp tree with fuzzification, and a bagging ensemble of 50 crisp trees using randomized splits. In the first two cases the errors of the crisp model tree are clearly visible. They are larger around the five split cut-points, $\frac{i}{12}$ where $i = 1, 2, \cdots 12$. Also, a discontinuity is visible in the first case, for $x = \frac{5}{12}$, where the predictions for $x < \frac{5}{12}$ are larger than the predictions of $x > \frac{5}{12}$. This discontinuity issue is fixed in the second case, however the error improvement of the fuzzified model tree is small. On the other hand, the bagging ensemble shows an improved fit to the nonlinear function, as compared to a single M5' model tree.

If we compare the fit of the crisp model tree, shown in the left part of Figure 4.8, to the fit of the soft model tree, shown in the left part of Figure 4.7, we can note that the performance of the crisp model tree is worse than the single Lolimot tree with the same number of local models/terminal nodes. This shows the advantage in terms of predictive power that the soft tree variants possess. This advantage is more clearly visible when the function to be fit contains "more nonlinearity", an example of which is the function chosen here. When comparing the errors of the ensembles, however, we can conclude that both ensembles produce similar results, the bagging of Lolimot trees being slightly better.

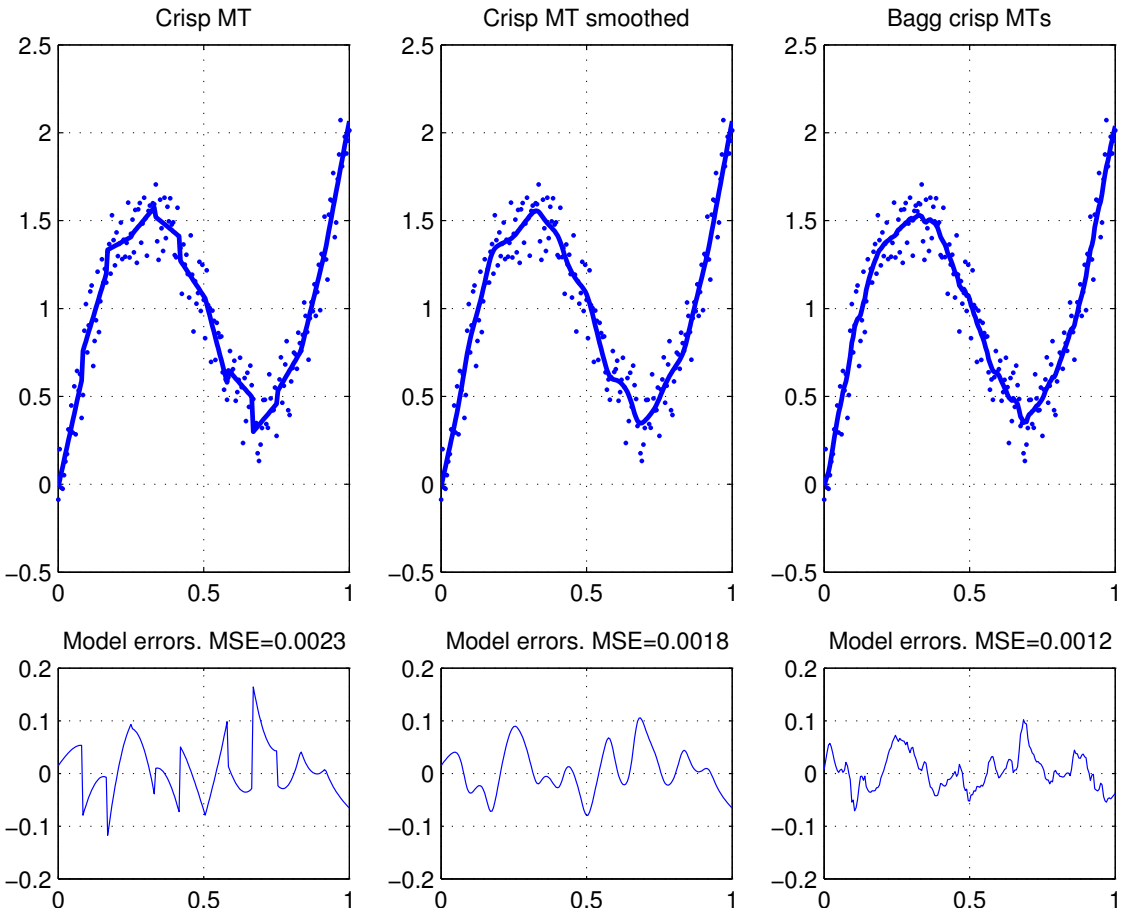


Figure 4.8: Performance of the crisp model tree approaches. A single M5' tree (left), a smoothed variant of the M5' tree (middle) and bagging of M5' trees (right). The M5' model tree has 12 terminal nodes. The bagging consists of 50 model trees with 12 terminal nodes. The lower panels show the approximation error $f(x) - \hat{f}(x)$. In both cases the modeling is performed by using data with 20% noise.

4.4.1 Derivatives of the Models

This analysis focuses on the derivatives of the model, using the same static function. The derivatives of the model play an important role to the model performance, as they determine the local linearizations of the nonlinear model of the system. Also, the correct fit of the model to the system derivatives would mean that acceptable OSA and simulation performance can be expected. The aim of the analysis is to show the correspondence of the derivatives of different model types to the true derivatives of the static function.

The derivative of the static nonlinear function defined in Eq. (4.16), with respect to x is:

$$\frac{df}{dx} = 2\pi \cos(2\pi x) + 2. \tag{4.17}$$

The derivatives of the models produced are calculated differently. The derivative of the crisp model tree is the slope coefficient of the corresponding local model. The derivative of a soft model tree is a weighted sum of the slopes of all local models (note that the weights add up to 1). The derivative of an ensemble, such as bagging, which produces the final prediction by averaging the predictions of the base models, is calculated by averaging the slopes of each base model.

The results in Figure 4.9 show that the fuzzy MT has a closer fit to the true function derivatives, as compared to a crisp MT. When comparing to the smoothed version, we can conclude that the fit to the true derivatives is similar. Both bagging variants produce models with closer fit to the true derivatives, however, the model derivatives show larger variation around the true function derivative. The large variation is probably due to the randomization of the cut-points, which is used to illustrate the ensembles.

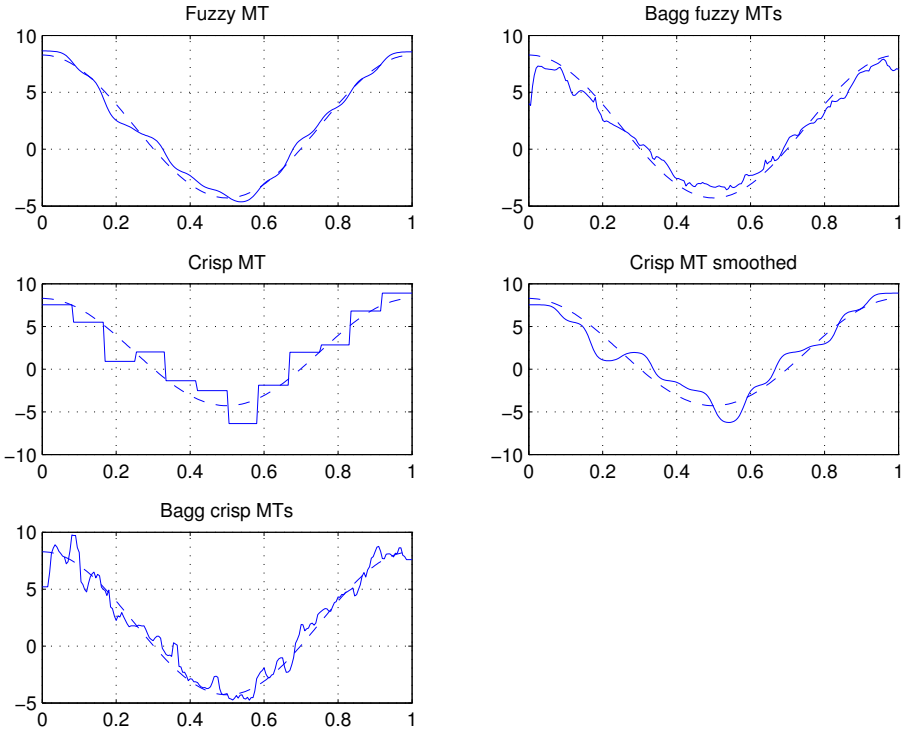


Figure 4.9: An illustration of the derivatives of the different models (solid lines) compared to the true derivative of the function (dashed line).

Chapter 5

Evaluation on Benchmark Machine Learning Regression Datasets

This chapter of the thesis would analyze the performance of regression trees, model trees and ensembles, for the tasks of single-target and multi-target regression. The datasets used are considered as benchmark machine learning datasets, and the results should provide some insight into the potential of the model tree ensembles. In more detail, the analysis in this chapter will consider three different tree formalisms: crisp regression trees, crisp model trees and soft model trees. Also, it will consider ensembles of all the three mentioned formalisms. The questions that it will try to provide answers to, are the following:

- How well do the different tree formalisms perform for single-target and multi-target regression?
- Do the ensembles of model trees offer an increase in performance over single model trees?
- How do forests compare to bagging, when using model trees as base models?
- What is the influence of the ensemble size on its performance?

All of the questions are going to be analyzed by using a group of 49 single-target datasets, and a group of 9 multi-target datasets. The evaluation would be performed by using cross-validation. The results would be analyzed by using statistical tests, which would determine whether the differences obtained are statistically significant.

The first question is going to be addressed by comparing the different tree variants on the two dataset groups. The measures that would be reported and analyzed consider the predictive performance, the model sizes, and the time required for learning. The second question is going to be addressed by analyzing the performance of bagging and forest ensembles. Recall that the forests differed from bagging due to the introduction of randomness in the tree building procedure. The performances of bagging and forests would be compared to that of a single tree. The third question would compare the two ensemble approaches of bagging and forests, and evaluate whether the randomization included in the forests is beneficial over the bagging ensemble. Finally, for the last question, ensembles of model trees using different sizes would be evaluated. The results here would be utilized for determination of a sensible value for the ensemble size parameter.

The rest of the chapter describes the single-target and multi-target datasets, used in the analysis. It also provides information regarding the preprocessing of the data. In the following it describes the experimental setup, the performance measures and the statistical tests used. Also, it provides information regarding the parameters evaluated for each of the algorithms. Finally, it presents the experimental results.

5.1 Datasets

The benchmark machine learning datasets for regression, that are used can be divided in two groups:

- single-target regression datasets,
- multi-target regression datasets.

The description of each group can be found in Tables 5.1 and 5.2, respectively. The first consists of 49 single-target datasets, while the second consists of 9 multi-target datasets. For the multi-target case, the total number of target variables is equal to 33. The datasets for the single-target regression task are taken from two repositories (Torgo, 2013; Asuncion & Newman, 2007), while the multi-target datasets are obtained from several sources, reported in the table along with the dataset information.

Table 5.1: The list of single-target regression datasets. The table reports the number of instances n, the number of attributes a, and the number of nominal attributes a_{nom}.

	Dataset	n	a	a_{nom}
01.	abalone	4177	9	1
02.	analcat	4052	8	5
03.	auto93	93	23	6
04.	autoMpg	398	8	3
05.	auto-price	159	16	1
06.	bank8FM	8192	9	0
07.	baseball	337	17	4
08.	baskball	96	5	0
09.	bodyfat	252	15	0
10.	breastTumor	286	10	8
11.	cal-housing	20640	9	0
12.	cholesterol	303	14	7
13.	cleveland	303	14	7
14.	cloud	108	7	2
15.	concrete	1030	9	0
16.	cpu	209	8	1
17.	cpu-act	8192	22	0
18.	dailyElectrEner	365	7	0
19.	delta-ailerons	7129	6	0
20.	delta-elevators	9517	7	0
21.	echoMonths	130	10	3
22.	electr-len-2.arff	1056	5	0
23.	fishcatch	158	8	2
24.	forestFiresPOR	517	13	1
25.	fruitfly	125	5	2
26.	housing	506	14	1
27.	hungarian	294	14	7
28.	kin8nm	8192	9	0
29.	laser	993	5	0
30.	lowbwt	189	10	7
31.	machine-cpu	209	7	0
32.	meta	528	22	2
33.	mortgage	1049	16	0
34.	pbc	418	19	8
35.	pharynx	195	12	10
36.	pol	15000	49	0
37.	puma8NH	8192	9	0
38.	pwLinear	200	11	0
39.	quake	2178	4	0
40.	sensory	576	12	11
41.	servo	167	5	4
42.	stock	950	10	0
43.	strike	625	7	1
44.	treasury	1049	16	0
45.	triazines	186	61	0
46.	veteran	137	8	4
47.	wankara	1609	10	0
48.	wisconsin	194	33	0
49.	wizmir	1461	10	0

5.1.1 Preprocessing

The preprocessing part deals with the missing values in the dataset and the nominal (discrete) attributes which have more than two possible values. The datasets have been preprocessed with the default preprocessing procedures of the M5' algorithm in the WEKA implementation.

Table 5.2: The list of multi-target regression datasets. The table reports the number of instances n, the number of attributes a, the number of nominal attributes a_{nom}, and the number of targets/outputs r.

	Dataset	n	a	a_{nom}	r
01.	Collembola (Kampichler, Džeroski, & Wieland, 2000)	393	50	8	3
02.	EDM (Karalič & Bratko, 1997)	154	18	0	2
03.	Forestry IRS (Stojanova, 2009)	2730	31	0	2
04.	Forestry SPOT (Stojanova, 2009)	2730	51	0	2
05.	Sigmea-real (Demšar, Debeljak, Lavigne, & Džeroski, 2005)	817	8	0	2
06.	Sigmea-simulated (Demšar, Debeljak, Lavigne, & Džeroski, 2005)	10368	13	2	2
07.	Solar-flare1 (Asuncion & Newman, 2007)	323	13	10	3
08.	Solar-flare2 (Asuncion & Newman, 2007)	1066	13	10	3
09.	Water quality (Džeroski, Demšar, & Grbović, 2000)	1060	30	0	14

The missing values in the dataset are replaced with modes and means from the training data. The discrete variables with more than two possible values are converted to several binary attributes. A discrete attribute with v possible values is converted into $v-1$ binary attributes, using the one-attribute-per-value approach (Breiman et al., 1984). For more details regarding the preprocessing of the data that the M5' implementation in WEKA performs, see (Hall et al., 2009).

5.2 Experimental Design

One of the goals of a sound experimental design is to determine a good estimate of the performance of the methods, on unseen data. This needs to be executed using a limited number of available data instances. In order to estimate the predictive performance of the obtained models, we employ the standard 10-fold cross-validation estimator. The data is split into $k = 10$ parts of approximately equal size. In the jth step, the part j is used as a test set, while the other $k-1$ parts are used as a training set. To allow for a fair comparison of the performance, for each of the selected methods we utilize exactly the same folds.

The optimal parameters of the methods, for each of the 10 folds, are selected using internal (nested) 5-fold cross validation. This means that for each of the 10 outer folds, we perform nested 5-fold cross-validation, which would determine the optimal set of parameters for the method in that fold (Witten & Frank, 2005). For each of the methods, several different parameter values are tried. The list of parameter values for each of the methods is reported in Table 5.3. It is worth noting that the Lolimot model tree algorithm considered was set up in order to evaluate the candidate splits according to the prediction error, as opposed to its default evaluation by using output error.

To test whether the obtained differences in performance are statistically significant given a single dataset, we apply the paired Student's t-test. This parametric test assumes that the population follows a normal distribution. The paired t-test is applied to the results of the methods on each of the 10 folds of the cross-validation procedure. In the case of multiple target variables, the differences are assessed and reported for each target variable independently.

To test the difference in performance of algorithms on all available datasets, we utilize the non-parametric Wilcoxon signed rank test (Wilcoxon, 1945). It is commonly used when the assumption of normality of the population cannot be made. For the predictive

performance, we utilize the error from the cross-validation procedure, i.e., the aggregated performance from each of the folds. The significance level used for both the paired t-test and the Wilcoxon test is 0.01. This means that the null hypothesis, which states that the difference between the two responses is zero, is rejected when the p-value is less than the significance level of 0.01.

Table 5.3: Method parameters considered for the experimental evaluation.

Method name	Parameter name	Values considered
M5' model tree (MT)	Post-pruning	T/F
	LS Regression: only M5' features	T/F
	LS Regression: M5' feature selection	T/F
	Smoothing of linear models	T/F
M5' regression tree (RT)	Post-pruning	T/F
Lolimot	Maximal num. iterations	30
Bagging of M5' MTs or RTs	Number of trees	100
Forest of M5' MTs or RTs	Number of trees	100
	Size of random subset of feat. att	[0.2; 0.4; 0.6; 0.8] *#features
Bagging Lolimot	Number of trees	100

5.2.1 Performance Measures

The experimental analysis evaluates the performance of the models learned, using different measures. The performance measures consider three aspects of the models: a) the predictive performance of the models, b) the time required for model learning and c) the size of the resulting model.

The predictive performance is measured in terms of root relative mean-squared error, or abbreviated RRMSE, calculated as:

$$RRMSE = \frac{\sqrt{\sum (y_i - \hat{y}_i)^2}}{\sqrt{\sum (y_i - \bar{y})^2}} \tag{5.1}$$

The tables in Section 5.3 report the aggregated RRMSE error of the 10 folds of the cross-validation procedure. The results of the multi-target regression report the RRMSE for each target separately.

The time required for model learning is reported in seconds, and it includes the total time required for the 10-fold cross-validation procedure. The size of the trees is reported as the number of terminal nodes (local models) of the tree. In a similar fashion, by size of an ensemble of trees, we denote the average number of terminal nodes in the trees of the ensemble. This notation is used in the whole chapter.

For all of the measures reported, a smaller value indicates better performance. This is also consistent in the statistical test reports. For example, a result of the t-test which reports the value 10:5 for the running time of algorithm variant A compared to B means that for 10 datasets algorithm A had a stat.sign.smaller running time than B. Also it states that for 5 other datasets, algorithm B had a stat.sign.smaller running time than A.

5.3 Experimental Results

This section reports the results of the empirical analysis, both on single-target and on multi-target regression tasks. It is organized as follows: First, the performance of model

trees and regression trees is evaluated on the benchmark machine learning datasets. Then, the model trees are compared to ensembles, and finally, the ensemble size is evaluated, by considering ensembles with a different number of trees.

5.3.1 Evaluating the Performance of Different Tree Learning Algorithms

The experimental evaluation would compare the performance of different types of trees and ensembles thereof. This part outlines the types of trees which are used, and the algorithms that built them.

The comparison includes regression trees and ensembles thereof, typically evaluated in the machine learning domain. The regression trees are built by using the M5' algorithm with the regression tree setting. Also, it compares crisp and fuzzy model trees. The former are built by using the M5' algorithm, while the latter are built by using the Lolimot algorithm. The crisp model trees are frequently included in machine learning comparative studies, as single trees. However, ensembles of crisp model trees, fuzzy model trees, and ensembles thereof are rarely used in machine learning studies. The motivation of the analysis is to evaluate the applicability of the last two in the machine learning domain. In summary, the types of models and the algorithms used in the evaluation, are as follows:

- crisp regression trees, built with the M5' algorithm (M5' RT, or only RT),
- crisp model trees, built with the M5' algorithm (M5' MT, or only MT),
- soft model trees, built with Lolimot (Lolimot).

In the following subsections, the different tree learning algorithms would be compared by first considering the single-target regression task, followed by an analysis of the methods for multi-target regression.

5.3.1.1 Single-target Regression

Here we report the performance of the regression tree approach and the two model tree approaches for the single-target regression tasks. The results of the predictive performance are summarized in Table 5.4. On the one hand, they show that there is no significant difference between the performance of M5' model trees and Lolimot trees. The t-test reports that the M5' model tree outperforms Lolimot in 5 of the 49 cases, while the Lolimot model shows better performance in 7 of the 49 cases. The Wilcoxon test detects no statistically significant difference at the 1% level, which is also supported by the markers placed mainly around the diagonal, in Figure 5.1. On the other hand, the comparison to regression trees shows that M5' model trees are a better performing approach, and this difference is statistically significant according to the Wilcoxon test.

Table 5.4: A statistical comparison of the predictive performance of model trees (MT), soft model trees (Lolimot), and regression trees (RT), for the task of single-target regression. A summary of Table A.1. The results reported in all tables compare the leftmost method, in this case M5' MT, to all of the other methods, by using paired comparisons.

M5' MT :	Lolimot	RT
t-test	5:7	21:0
w-test	0.384	0.000

Table 5.5: A statistical comparison of the model sizes and running times of model trees (MT), soft model trees (Lolimot), and regression trees (RT), for the task of single-target regression. A summary of Table A.2. The number of wins, denoted as "#wins" is reported in the first row in this and in the following tables with results for the size of the models and the running time. The values only summarize the number of datasets on which variant A had a smaller value than variant B, i.e., no statistical test is considered. The sum of the number of wins for the method tested and its alternative would always add up to the total number of datasets.

	Model size			Learning time (sec.)	
M5' MT :	Lolimot	RT	M5' MT :	Lolimot	RT
#wins	23:26	42:7		49:0	1:48
w-test	0.003	0.019		0.000	0.000

Considering the size of the trees, reported in Table 5.5, we can conclude that M5' builds larger trees than Lolimot, and the difference is statistically significant. Also, a RT is typically larger than an M5' model tree, however the difference is not statistically significant at the 1% level. Both of these results are expected, since one could expect that the soft model tree requires less local models to obtain the same predictive performance as compared to the crisp model tree. Also, in theory, a linear model of the model tree is able to replace a subtree of the regression tree, modeling a linear relationship, and lowering the number of local models.

In spite of the differences in sizes, the running times are the smallest for regression trees, and largest for the Lolimot method. This is expected for M5' MTs and M5' RTs, due to the same tree building phase, and the additional local model estimation performed by the former approach during the tree pruning phase. Also, it is expected when comparing M5' to Lolimot, mainly because of the lookahead step included in the Lolimot approach, which includes estimation and evaluation of local linear models.

5.3.1.2 Multi-target Regression

The results of the predictive performance on the multi-target regression tasks are summarized in Table 5.6. Similar to the single-target case, they suggest no difference between the performance of M5' and Lolimot, and a statistically significant difference between M5' model trees and regression trees.

The comparison results of an M5' MT to a RT, show that the former wins on 4 of the 9 multi-target datasets. Also, the Wilcoxon test reports this as a statistically significant difference, at the 1% level. More insight into the differences in performance are presented in Figure 5.2. The results there confirm that the model tree outperforms the regression tree on the multi-target regression task.

Table 5.6: A statistical comparison of the predictive performance of different tree learning algorithms for the task of multi-target regression. A summary of Table A.3.

M5' MT:	Lolimot	RT
t-test	0:0	4:0
w-test	0.150	0.006

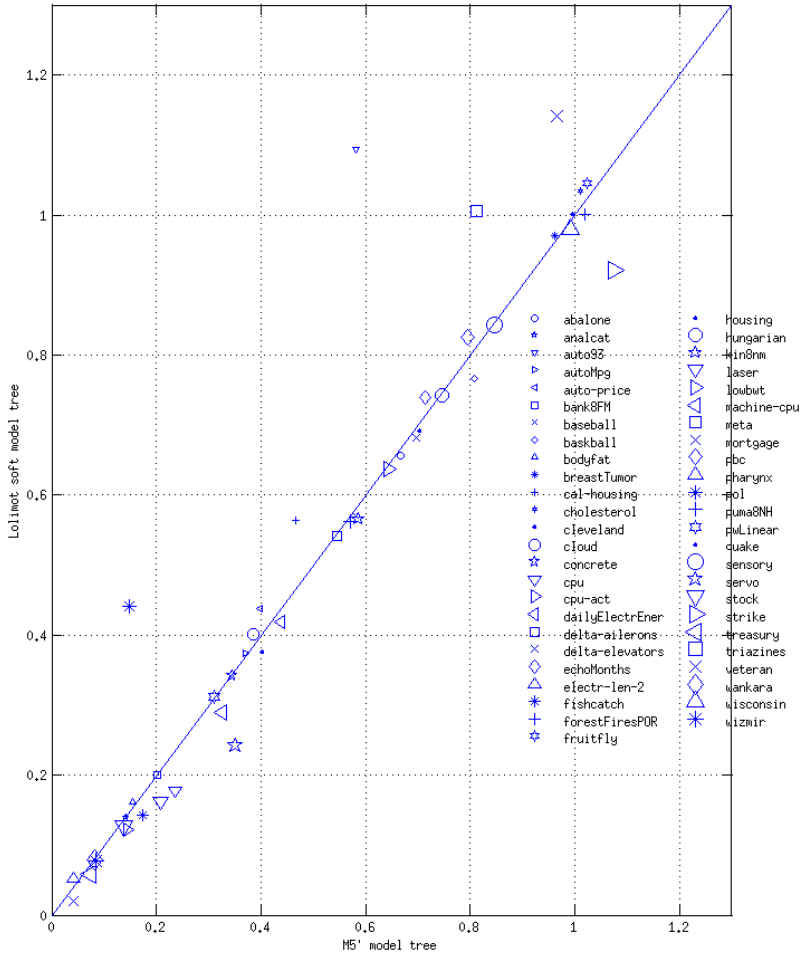


Figure 5.1: A comparison of the predictive performance of Model trees (MT) and soft model trees (Lolimot).

Table 5.7: A statistical comparison of the model sizes and running times of different tree learning algorithms for the task of multi-target regression. A summary of Table A.4.

	Model size			Learning time	
M5' MT:	Lolimot	RT	M5' MT:	Lolimot	RT
#wins	3:6	8:1		9:0	0:9
w-test	0.039	0.875		0.004	0.004

The results for the model sizes in Table 5.7 show that the M5' MTs and RTs have similar sizes for most of the datasets. The w-test produces quite a high p-value, indicating that there is very little evidence of differently-sized models. The comparison of M5' MTs to Lolimot is in favor of the latter, but the difference is not statistically significant.

5.3.2 Comparing Model Trees to Ensembles

This subsection compares the performance of ensembles of different types of trees, to the performance of an M5' model tree. The decision to consider only the M5' model tree here was influenced by the results from the previous subsection, where the M5' model tree and Lolimot both showed acceptable performance, however, the learning times of the M5'

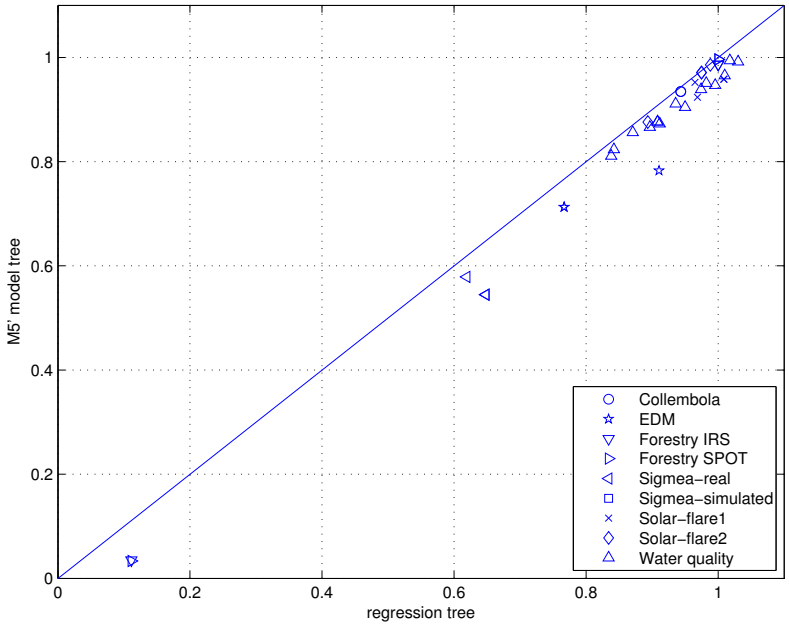


Figure 5.2: A comparison of the predictive performance of an M5' model tree to a regression tree for the task of multi-target regression. Each marker represents one target variable of the corresponding multi-target dataset.

algorithm were substantially smaller.

The ensembles considered in this subsection are built using the three types of base models, discussed earlier. Also, the ensembles are built by using both bagging and forest approaches, with the exception of the Lolimot method, where only bagging is considered. In more detail, the method variants considered here, and their abbreviations are: forests of M5' model trees (FMT), bagging of M5' model trees (BMT), a single M5' model tree (MT), forests of M5' regression trees (FRT), bagging of M5' regression trees (BRT) and bagging of soft Lolimot trees (BL). All ensembles analyzed in this part consist of a fixed number of 100 trees. This analysis first compares the methods for the single-target regression tasks, and uses the datasets listed in Table 5.1. Then it considers the multi-target regression tasks and the datasets listed in Table 5.2.

5.3.2.1 Single-target Regression

The results of the ensembles for the single-target regression task are summarized in Table 5.8. The statistical tests show that the difference in predictive performance of forests of M5' model trees is statistically significant when compared to a single M5' model tree, and when compared to each of the ensemble variants built using regression trees or soft Lolimot model trees. The difference of forests of M5' model trees to bagging of M5' model trees is not statistically significant at the 1% level, according to the Wilcoxon test. Also, the t-test reveals that the number of datasets on which the former is significantly better than the later, is only 7, out of 49.

Table 5.8: A statistical comparison of the predictive performance of the ensemble approaches for single-target regression. All ensembles consist of 100 trees. Summary of Table A.5.

FMT :	BMT	MT	FRT	BRT	BL
t-test	7:1	16:0	13:0	14:0	12:1
w-test	0.028	0.000	0.001	0.000	0.003

Table 5.9: A statistical comparison of the model sizes and running times. Summary of Table A.6.

	Model size					Learning time (sec.)				
FMT :	BMT	MT	FRT	BRT	BL	BMT	MT	FRT	BRT	BL
#wins	18:31	22:27	37:12	28:21	11:38	2:47	0:49	0:49	0:49	2:47
w-test	0.001	0.236	0.012	0.462	0.000	0.000	0.000	0.000	0.000	0.000

A comparison of the predictive performance of forests of model trees to a single model tree is shown in Figure 5.3. The results in the figure show that the most of the markers are below the diagonal, which indicates that a forest of M5' model trees provides an improvement in predictive performance over a single M5' model tree.

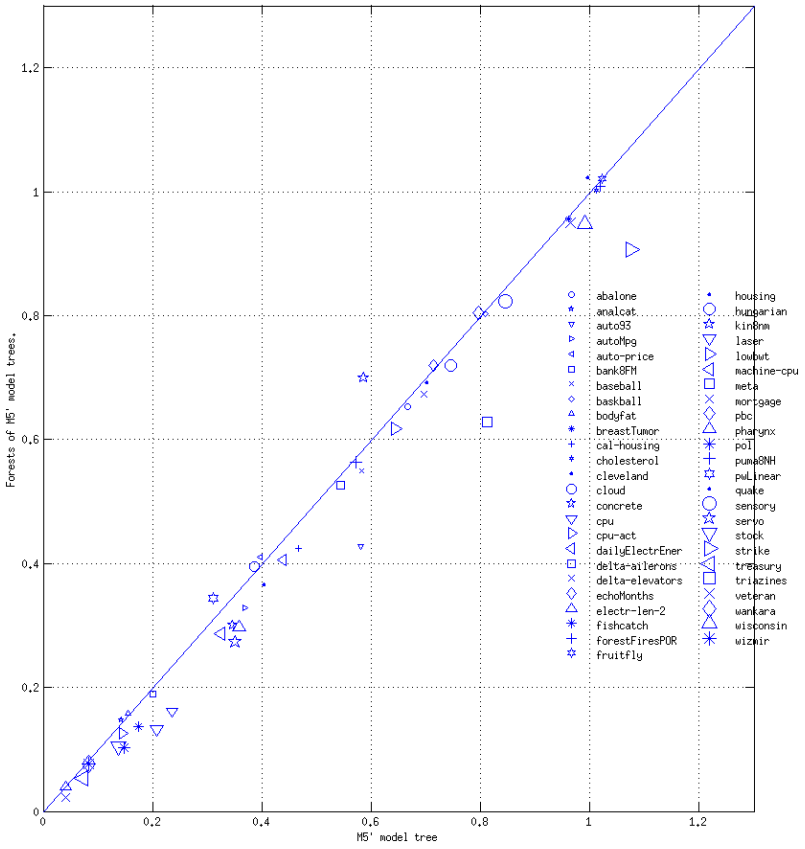


Figure 5.3: A comparison of the predictive performance of a single M5' model tree to that of forest ensemble of M5' model trees.

Regarding the sizes of the trees, summarized in Table 5.9, the results suggest that bagging of M5' model trees builds smaller trees than forests of M5' model trees. Also, forests of M5' model trees build smaller trees than forests of regression trees, which is somewhat expected. In theory, the linear models can potentially replace a subtree of the regression tree consisting of several splits, which forms a piecewise constant model. The results for the soft model trees show that the bagging of Lolimot trees approach creates trees with less local models than the forests of model trees. This can also be considered as expected, since the soft model tree formalism would require less local models than the crisp one, to achieve comparable accuracy on a smooth function approximation task.

5.3.2.2 Multi-target Regression

The results of the multi-target regression analysis for the different types of ensembles are shown in Table 5.10, which is a summary of Table A.7. The results reveal that the forests of M5' model trees show improvement over bagging of M5' model trees and a single M5' model tree, while they have similar performance to the regression tree and Lolimot variants. The Wilcoxon test shows that the difference to the bagging of M5' model trees and a single M5' model tree is significant at the 1% level. The performance of forests and a single M5' model tree is also presented visually, in Figure 5.4. The most of the markers visible are below the diagonal, which suggests that the forests improve over the single tree. However, the markers for two of the datasets are not visible, as their errors were out of the scope of the figure.

Table 5.10: A statistical comparison of the predictive performance. Summary of Table A.7.

FMT :	BMT	MT	FRT	BRT	BL
t-test	1:0	3:0	3:0	2:0	4:0
w-test	0.001	0.001	0.837	0.102	0.027

Table 5.11: A statistical comparison of the model sizes and running times. Summary of Table A.8.

	Model size					Learning time (sec.)					
FMT :	BMT	MT	FRT	BRT	BL	FMT :	BMT	MT	FRT	BRT	BL
#wins	4:5	4:5	9:0	5:4	2:7		1:8	0:9	0:9	0:9	0:9
w-test	0.078	0.375	0.250	0.688	0.016		0.020	0.004	0.004	0.004	0.004

A detailed look at the results shows large errors obtained by the three ensemble variants using model trees, on the Sigmea simulated dataset. The large errors values are evident for all of the targets in this dataset, and they convey that some of the model trees in the ensemble have incorrect local model coefficients. In more detail, Table A.7 reveals that all three variants using M5' model trees, i.e., a forests of M5', bagging of M5' and a single M5' model tree, have large errors for both targets of the Sigmea simulated dataset. However, the t-test outcome states an equal performance to bagging of regression trees and of Lolimot trees. Further analysis of the results for this dataset show that for several of the folds the model tree and ensembles thereof contain incorrect local models, and the error for the folds in question is quite large. The model errors corresponding to these folds also increase the final cross-validated RRMSE error measure. For the other folds, the

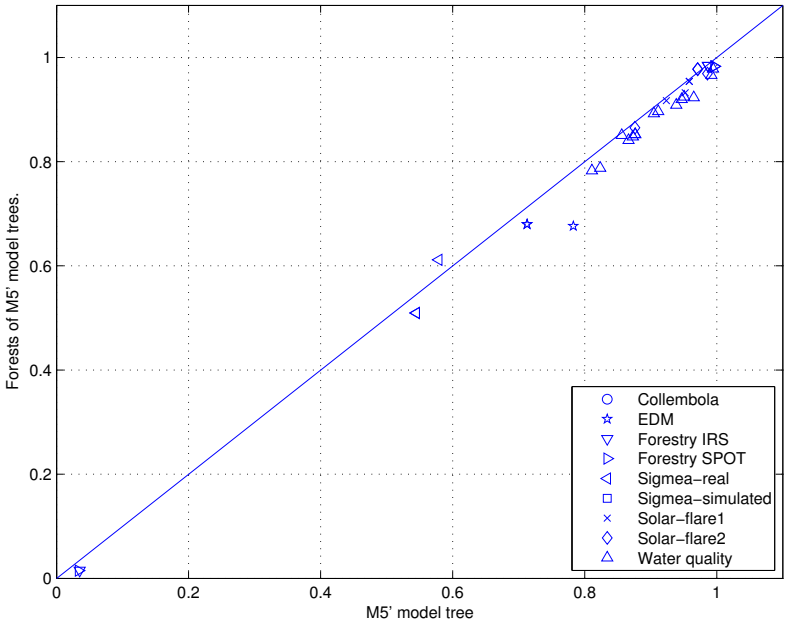


Figure 5.4: A comparison of the predictive performance of a single M5' model tree to that of forest of M5' model trees for the task of multi-target regression.

errors of all three M5' model tree-based methods are comparable to the methods based on regression trees and Lolimot trees.

The incorrect local models learned for this dataset might be due to the small number of training points in some of the partitions. One possible solution to this problem is the "capping", as mentioned in the work of Pfahringer (2011). Using this approach, the model tree predictions are modified, so that they are in some pre-defined range. Overly large or small model predictions would be replaced by the minimum or maximum of the corresponding target variable in the training set.

Also, for the Collembola dataset, all variants based on model trees overprune and build rather small models consisting mostly of only one local model. This is the reason why both approaches based on regression trees have smaller errors on this dataset. In summary, the results suggest that the overpruning issue and the incorrect local models issue could reduce the applicability of model trees and ensembles thereof, for multi-target regression.

5.3.3 Ensemble Size

This subsection presents the analysis of the ensemble size on its performance, on the task of single-target regression. It considers the forests of M5' model trees, which proved to be the most successful ensemble approach of the ones compared earlier.

The results in Table 5.12 summarize the results when considering forests of 100, 50 and 25 model trees. The outcomes of the statistical tests show that there is a significant difference in performance between forests with 100 M5' model trees and forests with 25 trees. According to the Wilcoxon test, the difference is significant at the 1% level, while the t-test shows a significant difference for only two multi-target datasets. Additionally, the prediction performance results visualized in Figure 5.5 show that the improvement in performance by using 100 trees over 25 trees, is rather small.

Table 5.12: A statistical comparison of the predictive performance of forests of M5' model trees with a different number of trees, for the task of single-target regression. Summary of Table A.9.

FMT(100) :	FMT(50)	FMT(25)
t-test	0:1	2:0
w-test	0.064	0.000

Table 5.13: A statistical comparison of the model sizes and running times of forests of M5' model trees with a different number of trees. Summary of Table A.10.

	Model size			Learning time (sec.)	
FMT(100) :	FMT(50)	FMT(25)	FMT(100) :	FMT(50)	FMT(25)
#wins	29:20	25:24		8:41	0:49
w-test	0.885	0.770		0.000	0.000

The comparison of the model tree sizes in each of the forest variants, shown in Table 5.13, suggests that all three variants build model trees with comparable sizes. In other words, the results do not offer enough evidence to reject the null hypothesis of same-sized model trees. As expected, the running times are in favor of the ensemble with 25 model trees.

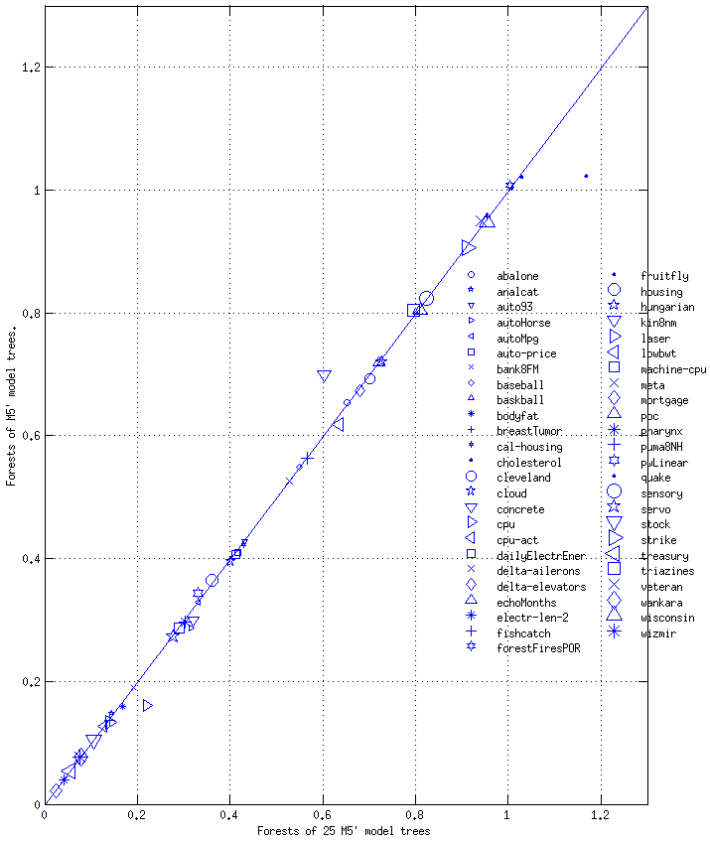


Figure 5.5: A comparison of the predictive performance of forests of M5' model trees with 100 and 25 model trees.

5.3.4 Summary

In summary, the empirical analysis showed that the model trees outperform the regression trees, and their size is also smaller. It also showed that the forests of M5' model trees improve the predictive performance over a single model tree. Additionally, the comparison of the forests of M5' model trees to the regression tree ensembles showed an improvement only in the single-target case. In the following, we outline some of the other conclusions which this analysis provides.

The predictive performance comparison of the single trees showed that there was no statistically significant difference between Lolimot and M5' MTs. Also, the M5' MTs outperformed the RTs, and the difference was statistically significant. The differences for the latter comparison were visible both in the single-target, as well as the multi-target analysis. Such results were expected, since the M5' and Lolimot model trees are a more powerful modeling formalism than regression trees.

Regarding the comparison of the sizes of trees, in terms of the number of terminal nodes, the results showed that Lolimot built smaller or equal trees as the M5' MT algorithm. This might be affected, however, by the limit of 30 terminal nodes in the Lolimot tree. The comparison of M5' MTs to RTs showed that the model trees require less terminal nodes for some datasets and a comparable number of terminal nodes for others. The statistical test did not detect a significant difference between these two. Since both are built using an identical tree building phase, the difference in size is due to the different amount of pruning. As expected here, the linear models in the model trees allow the pruning procedure to perform more reduction of the tree size. This was visible both in the comparison of the single trees in Subsection 5.3.1, and in the comparisons of ensembles, where the sizes of trees included in an ensemble were analyzed, cf. Subsection 5.3.2.

This chapter also tried to answer the question whether ensembles of model trees improve the predictive performance over a single model tree. A conclusion regarding this can be made from both Figure 5.3 and 5.4: The forests of M5' model trees increase the predictive performance over a single M5' model tree. The difference is statistically significant for both the single-target, and the multi-target regression, at the 1% level. However, the multi-target analysis showed that the M5' multi-target model tree algorithm needs to overcome two potential issues in the ensemble setting. The first is the identification of incorrect local models, which is due to the small sample of data points, while the second is that the pruning procedure can reduce the tree to a single local model. The issues appeared in 2 and 3 of the 9 multi-target datasets, respectively.

To overcome the issues it might be beneficial to limit, i.e., cap, the predictions of the model tree in the ensemble setting. The regression tree approaches do not have this problem, since they only learn a constant model in the terminal nodes, i.e., only fit one parameter (intercept) using the potentially small number of training points. On the other hand, the model trees use the same training points to fit $p + 1$ parameters.

The analysis included a comparison of the forests of M5' model trees to bagging. It concluded that the forests either improve the predictive performance over bagging, or in the worst case show similar results. On the one hand, the forests have the advantage of faster learning than bagging, since the tree algorithm needs to evaluate a smaller amount of candidate splits. On the other hand, the forests require that the size of random subset of features be tuned. Regarding the comparison of forests with a different number of trees, it was shown that there is a significant improvement when considering a forest of 100 trees, over a forest of 25 trees.

The comparison of ensembles of M5' model trees to ensembles of Lolimot showed that forests of M5' model trees significantly improved the predictive performance over a bagging of the same number of Lolimot trees, for the single-target regression. For the multi-target

regression the improvement was not significant, however differences in performance in favor of the first were again visible. The running times, were, as expected, in favor of the crisp model tree ensembles.

It is worth noting that, as future work, the analysis could be extended by including related methods that build soft or fuzzy regression trees. They could potentially build competitive models, however the algorithms that learn such trees are typically more computationally expensive than the M5' algorithm used here. Examples of soft regression trees, or in more general terms fuzzy decision trees, include the approaches of Olaru and Wehenkel (2003) and Suarez and Lutsko (1999).

Chapter 6

Evaluation for Modeling Dynamic Systems

The empirical evaluation in this chapter will try to answer several questions regarding the application of model trees and ensembles for modeling nonlinear dynamic systems in discrete time. To achieve this, it is going to consider seven dynamic system case studies that will be used in the empirical evaluation. In the following, a summary of all available data would be presented, along with the details of the experimental procedure, the methods selected for comparison, as well as their parameters. Finally, the results of the experiments would be presented, starting with the analysis of the model tree algorithm modifications and finishing with the evaluation of the model tree ensembles for single-output and multi-output modeling of nonlinear dynamic systems.

The questions that this evaluation will try to answer consider the using of crisp and soft model trees for modeling dynamic systems. This chapter will present the results of the evaluation of the proposed modifications to the base learning algorithms M5', which produces crisp model trees, and Lolimot, which produces soft model trees. It will also empirically analyze the performance of ensembles, built using the two model tree algorithms. In particular it will try to provide answers to the following questions:

- Do the proposed modifications to the model tree learning algorithms improve the performance for modeling dynamic systems?
- Is the multi-output modeling beneficial, as compared to single-output modeling?
- Do the proposed ensemble methods improve over the performance of single model trees?
- Are the two different model tree algorithms resilient to noise?
- How does the ensemble of model trees compare to selected, frequently used methods for identification of dynamic systems?

The first question is going to be addressed by evaluating each of the proposed modifications to the model tree learning algorithms, by using several case studies which include measured and synthetic data of nonlinear dynamic systems. For the second question two types of models would be compared: a set of single-output models, where each model predicts one output variable, to a single multi-output model, which predicts all output variables simultaneously. The comparison would be performed by performing simulation, which assumes using predicted values for all output variables, and where the error accumulation could easily sort out the incorrect models. The third and fourth questions

would be addressed by performing empirical comparisons of model trees and ensembles of models trees, built on data from the same case studies. Recall that for some of the case studies measured data were available, and these already contain certain amounts of noise. On the other hand for the synthetic datasets, we add noise to the output variables. The last question is going to be addressed by comparing the ensembles to the frequently used feed-forward Neural Networks and the hybrid neuro-fuzzy approach ANFIS.

6.1 Dynamic System Case Studies

We take into consideration seven dynamic system case studies from the areas of industrial engineering and mechanical engineering. One of the case studies presents a real world scenario of an experiment performed using a semi-industrial process plant at the Jožef Stefan Institute, i.e., the gas-liquid separator. Three other case studies, namely the Continuous-stirred Tank Reactor (CSTR), Steam Generator and the Winding process, are described and their data are published in the Daisy repository (De Moor, 2013). One of the case studies, concerning an anthropomorphic robot arm, is described and published by an independent source. Finally, two of the case studies, namely, pH Neutralization, and Narendra, are synthetic.

For each of the case studies, we also provide information regarding the input signals and the sampling time. The reasons for providing this information along with the description of the case studies were outlined in Section 2.1. Namely, the choice of the input signals during the data gathering procedure determines the distribution of the data points. Several different types of input signals exist, with two possible types being step-like (e.g., pseudo-random binary) and sine signals with different frequencies. The different types of excitation signals have a huge impact on the distribution of the data points that would be used for training and the expected accuracy of the model. In the remainder of this subsection we describe the dynamic system case studies which we use to evaluate our methodology.

6.1.1 Case Study: Continuous-stirred Rank Reactor

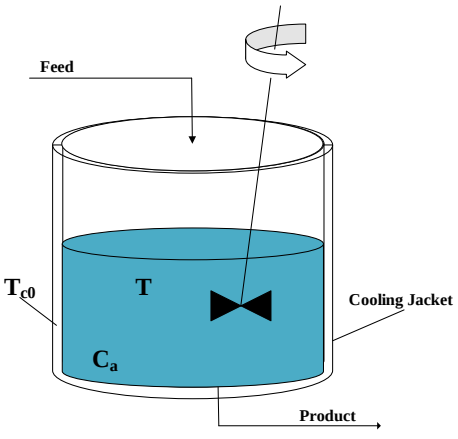


Figure 6.1: A diagram of the continuous-stirred tank reactor.

This case study concerns the well-known continuous-stirred tank reactor (CSTR). The CSTR process (Espinosa & Vandewalle, 1999; Lightbody & Irwin, 1997), depicted in Figure 6.1, describes a reaction of two products which are mixed. The products react and generate a compound A, whose concentration is $C_a(t)$. The temperature of the mixture is $T(t)$. This

exothermic reaction is controlled by introducing a coolant, whose flow rate is $q_c(t)$. The differential equations which describe the process are:

$$\dot{C}_a(t) = \frac{q}{v}(C_{a0} - C_a(t)) - k_0 C_a(t) e^{-\frac{E}{RT(t)}} \tag{6.1}$$

$$\begin{aligned} \dot{T}(t) &= \frac{q}{v}(T_0 - T(t)) - k_1 C_a(t) e^{-\frac{E}{RT(t)}} \\ &\quad + k_2 q_c(t)(1 - e^{-\frac{k_3}{q_c(t)}})(T_{c0} - T(t)) \end{aligned} \tag{6.2}$$

The modeling problem has one input variable (q_c) and two output variables (C_a and T). The numerical values for the other parameters of the model are given by Lightbody and Irwin (1997), Appendix A. The data that are used in this thesis are obtained from the Daisy repository (De Moor, 2013), where it is stated that the sampling time used to obtain the data T_s is 6 s. The number of data points is 7500, where the first 5000 are used as training points, and the last 2500 as testing points (Espinosa & Vandewalle, 1999). The input-output data of the test set are depicted in Figure 6.2 and denoted as CSTR.

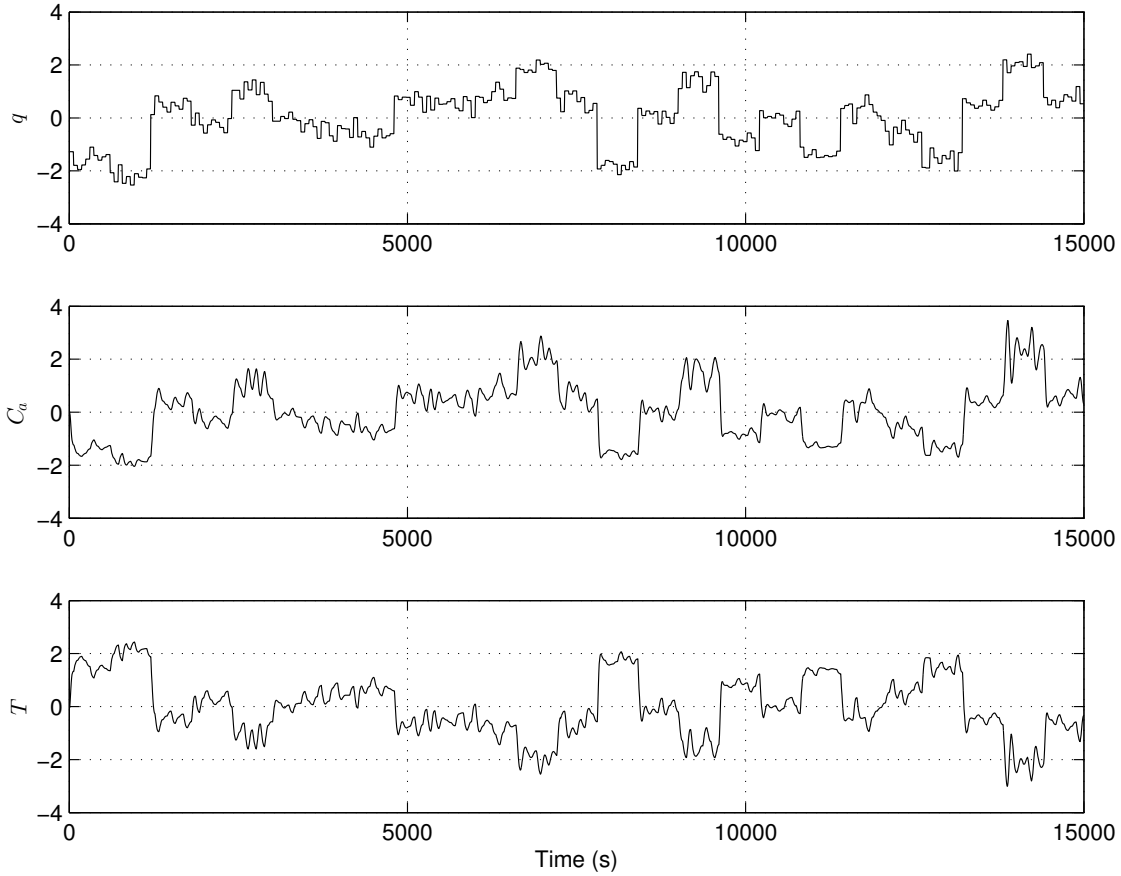


Figure 6.2: Normalized input-output data of the CSTR dynamic system. Data used for testing.

As Figure 6.2 shows, the CSTR data originating from the repository do not contain noise. In the analysis, we would also like to address the noise issue and to obtain insights into the noise-tolerance of the model tree and ensemble methods. For this purpose we create an additional version of the dataset, denoted as CSTR', with added noise. We added white

noise with mean zero and standard deviation of 20% of the standard deviation of the output variables. The noise was added only to the output variables in the training set, i.e., to all corresponding lagged variables in the set of features, as well as in the output variables (targets) of the training set. No noise was added to the test data.

6.1.2 Case Study: Gas-liquid Separator

The system being modeled in this case study is a unit for the separation of gas from liquid (Kocijan & Likar, 2008). The separator unit is a semi-industrial process plant which belongs to a larger pilot plant, residing at the Jožef Stefan Institute. A scheme of the structure of the plant is given in Figure 6.3.

The purpose of the modeled system is to capture flue gases under low pressure from the effluent channels using a water flow, cool the gases down, and supply them with increased pressure to other parts of the pilot plant. The flue gases coming from the effluent channels are absorbed by the water flow into the water circulation pipe through the injector I_1. The flow of water is generated by the water ring pump (P_1), whose speed is kept constant. The pump feeds the gas-water mixture into the tank T_1, where the gas is separated from the water. The accumulated gases in the tank form a kind of a pressurized gas 'cushion'. Due to this pressure, the flue gases are blown out from the tank into the neutralization unit, while on the other hand, the water is forced by the 'cushion' to circulate back to the reservoir. The water quantity in the circuit is constant.

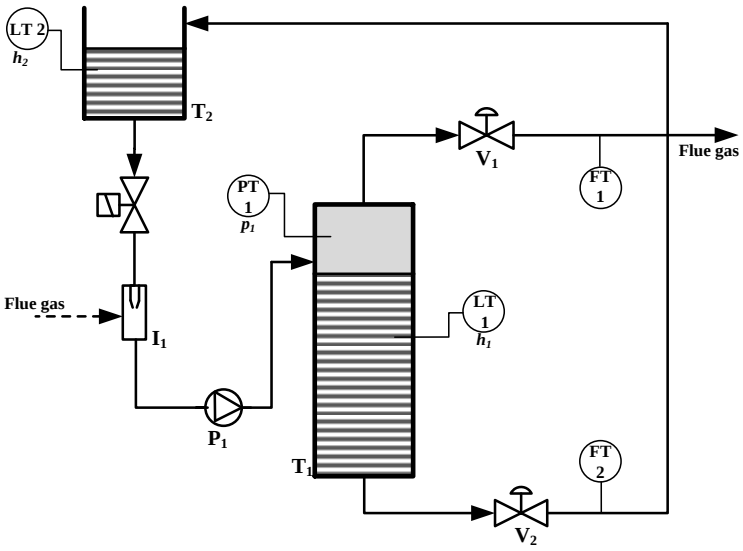


Figure 6.3: A schematic diagram of the semi-industrial process plant.

The first-principles model of the system is a set of differential equations. The variable p_1 is the relative air pressure in the tank T_1, the variable h_1 is the liquid level of the tank T_1, while u_1 and u_2 are command signals for the valves V_1 and V_2 respectively. The differential equation for the air pressure variable p_1 has the form:

$$\frac{dp_1}{dt} = f_a(h_1)[\alpha_1 + \alpha_2 p_1 + \alpha_3 p_1^2 + f_b(u_1)\sqrt{p_1} + f_c(u_2)\sqrt{(p_1 + \alpha_4 + \alpha_5 h_1)}] \tag{6.3}$$

where the values α_i are constants, while $f_a(h_1), f_b(u_1)$ and $f_c(u_2)$ are functions of the corresponding variables. $f_a(h_1)$ is a rational function of h_1, while $f_b(u_1)$ and $f_c(u_2)$ are

the valve characteristics (exponential functions of u_1 and u_2 respectively). The differential equation for the variable which denotes the liquid level of the tank, h_1, is:

$$\frac{dh_1}{dt} = \alpha_6 + f_c(u_2)\sqrt{(p_1 + \alpha_4 + \alpha_5 h_1)} \tag{6.4}$$

where the value α_6 is a constant. The details of the model are given by Kocijan and Likar (2008).

The aim of the system identification in this case study is to build a model for predicting the value of the pressure variable p_1, from lagged values of itself, as well as lagged values of the input variables. The sampling time selected was 20 s, same as in the work of Kocijan and Grancharova (2010). The training and the testing data both consist of 733 input-output data points, shown in Figure 6.4, and are disturbed by intrinsic measurement noise. The optimal lag was chosen by considering lag values from 1 to 3. For illustration, for a lag of 1, the system identification problem is transformed to the following regression problem:

$$p_1(k) = f_1(p_1(k-1), u_1(k-1), u_2(k-1), h_1(k-1)) \tag{6.5}$$
$$h_1(k) = f_2(p_1(k-1), u_1(k-1), u_2(k-1), h_1(k-1)) \tag{6.6}$$

of fitting the static nonlinear functions f_1 and f_2.

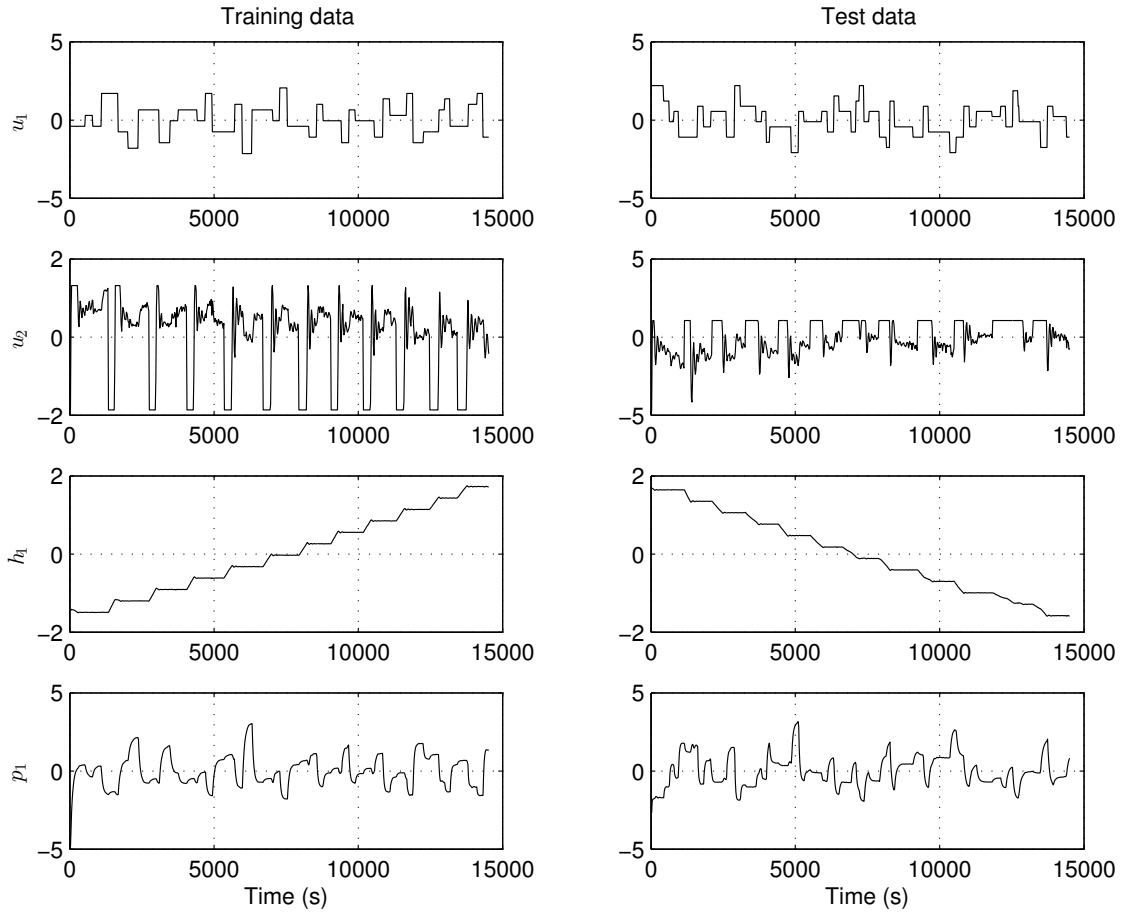


Figure 6.4: Input-output data for identification of the gas-liquid separator system. Detrended identification data are shown in the left and detrended validation data in the right four panels.

6.1.3 Case Study: Narendra System

This case study considers the synthetic Narendra nonlinear dynamic system (Narendra & Parthasarathy, 1990). This dynamic system is composed of an input variable u and a system variable y, which are connected by the following relation

$$y(k+1) = \frac{1}{1+y(k)^2} + u(k)^3. \tag{6.7}$$

The identification data shown in Figure 6.5 consists of 2000 data points for training and 2000 for testing, generated by using a different input signal u, with the same properties. In the experimental analysis we consider two versions of the data, one without and one with added noise. We added white noise with standard deviation of 20 % of the output variable deviation. The noise was added by following an identical procedure, as with the CSTR system, described above. The version of the data without noise is denoted by Narendra, while the version with noise is denoted by Narendra' in the empirical analysis.

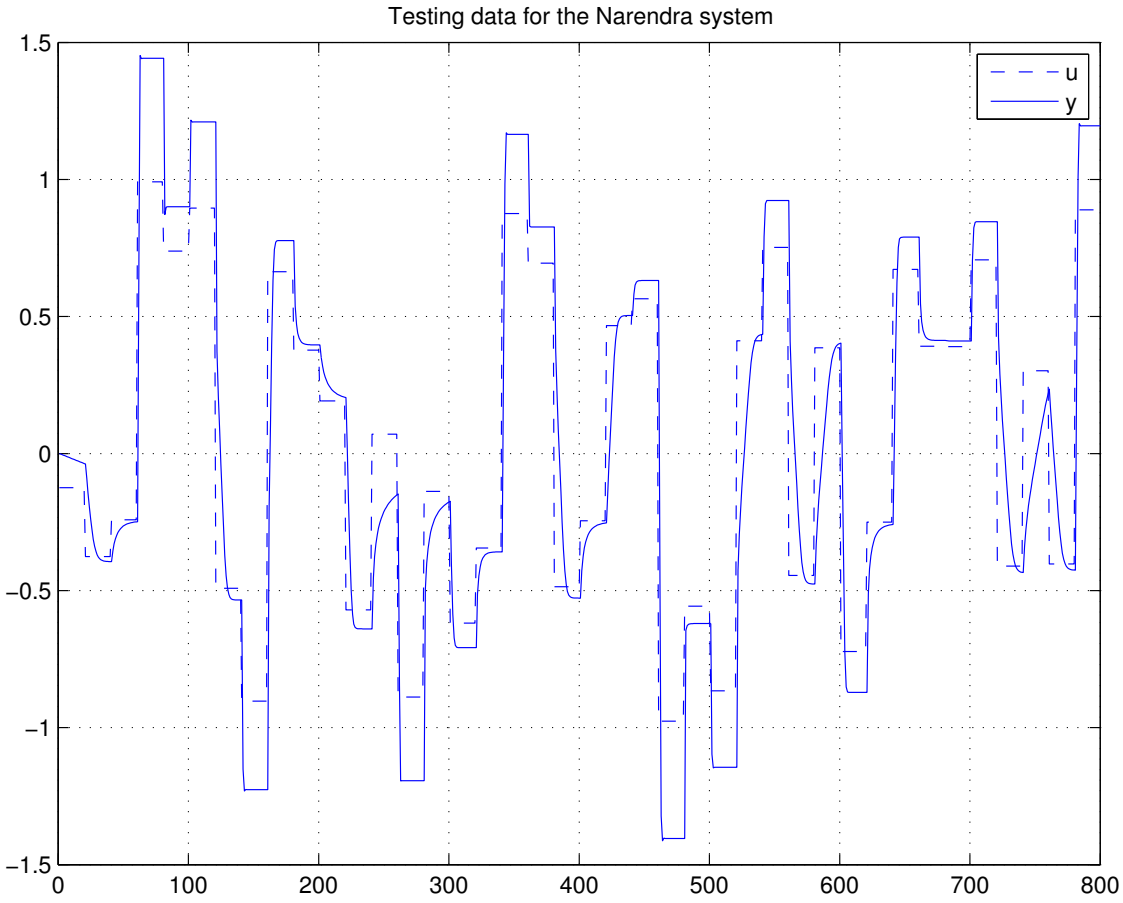


Figure 6.5: Input-output data for identification of the Narendra system. Testing data are shown, up to time step 800.

6.1.4 Case Study: pH Neutralization

The control of alkalinity (pH) is common in biotechnological industries and chemical processes. The topic of this case study is the identification of the pH neutralization process, which exhibits severe nonlinear behavior (Henson & Seborg, 1994; Kocijan & Petelin, 2011). What follows is a short description of the process itself, the equations governing the

process and the synthetic data generated from this model of the pH neutralization process, which are used for the task of system identification.

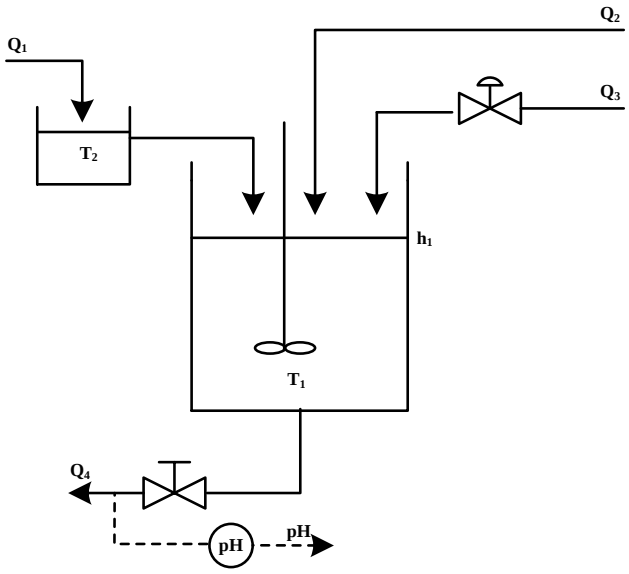


Figure 6.6: A schematic diagram of the pH neutralization system.

The pH neutralization system, described in detail by Henson and Seborg (1994), consists of an acid stream Q_1, a buffer stream Q_2, and a base stream Q_3 that are mixed in a tank T_1. Before mixing takes place, the acid stream Q_1 enters another tank T_2. The measured variable is the effluent pH, which is controlled by manipulating the flow rate of the base stream Q_3. The flow rates of the acid and buffer streams are taken to be constant. A schematic diagram of the system is shown in Figure 6.6.

A model of this dynamic system is derived by Henson and Seborg (1994), which contains the following state, input and output variables:

$$\mathbf{x} = [W_{a4}\ W_{b4}\ h_1]^T,\ u = Q_3,\ y = pH \tag{6.8}$$

where W_{a4} and W_{b4} are the effluent reaction invariants and h_1 is the liquid level of tank T_1. Also, it is assumed for the state variable h_1 that a controller has already been designed to keep its level at a nominal value of $h_1' = 14cm$ by manipulating the exit flow rate Q_4. The state-space model obtained has the form:

$$\dot{\mathbf{x}} = f(\mathbf{x}) + g(\mathbf{x})u \tag{6.9}$$

$$c(\mathbf{x}, y) = 0 \tag{6.10}$$

where $f(\mathbf{x})$ and $g(\mathbf{x})$ are nonlinear functions of the state vector $\mathbf{x}$, while $c(\mathbf{x}, y)$ is a nonlinear function which is a part of the implicit output equation (Eq. (6.10)). In the analysis, we consider four variants of the pH data, both with and without noise and generated by two different input signals. In all four variants the sampling time selected was 25 s, same as in the work of Kocijan and Petelin (2011).

In *the first two dataset variants* the input variable u changed its value every 500 s, each time being set to a value generated by using a uniform random distribution. The input-output data used, shown in Figure 6.7, consist of 320 data points for identification and 320 data points for validation. The first variant is denoted as pH_A.

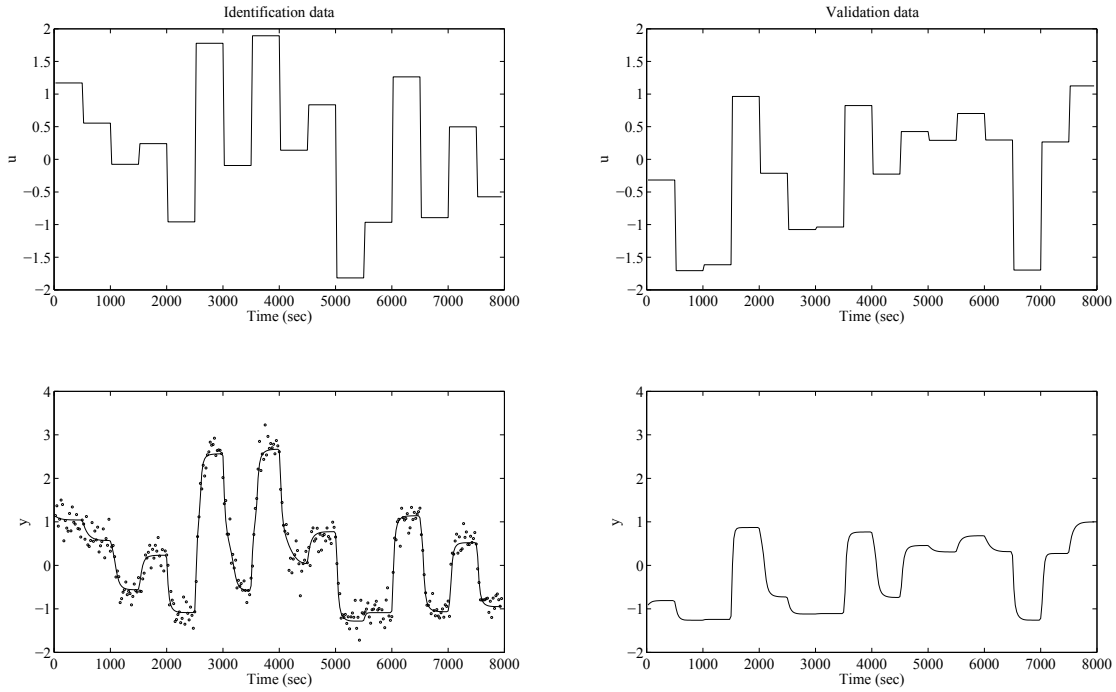


Figure 6.7: Input-output data for identification of the pH_A (and pH'_A) system; detrended identification data (left) and detrended validation data (right). The bottom left panel shows both the non-noisy data (solid line) and the data with 20% noise (dots).

To evaluate the resilience of the identification methods to noise, another variant of the data was considered, denoted as pH'_A, where white noise was added to the output (system) variable only in the identification data. The standard deviation of the added white noise was 20% of the output variable's standard deviation. The bottom left panel in Figure 6.7 shows the identification data with 20% noise. The validation data were not disturbed by white noise.

The third and the fourth dataset variants, denoted as pH_B and pH'_B consist of 400 data points for identification and 400 for validation. In this case the input variable u has considerably different dynamics, and the input signal u changed its value in every second time step, as shown in Figure 6.8. We use the notation pH_B to refer to the variant which does not contain noise, while pH'_B for the variant with 20% noise added, the same way as for pH'_A. The experimental procedure required a determination of the optimal dynamic order (lag) of the variables in the pH system. The values for the lag that we chose to evaluate for the four variants of the dataset ranged from 1 to 4.

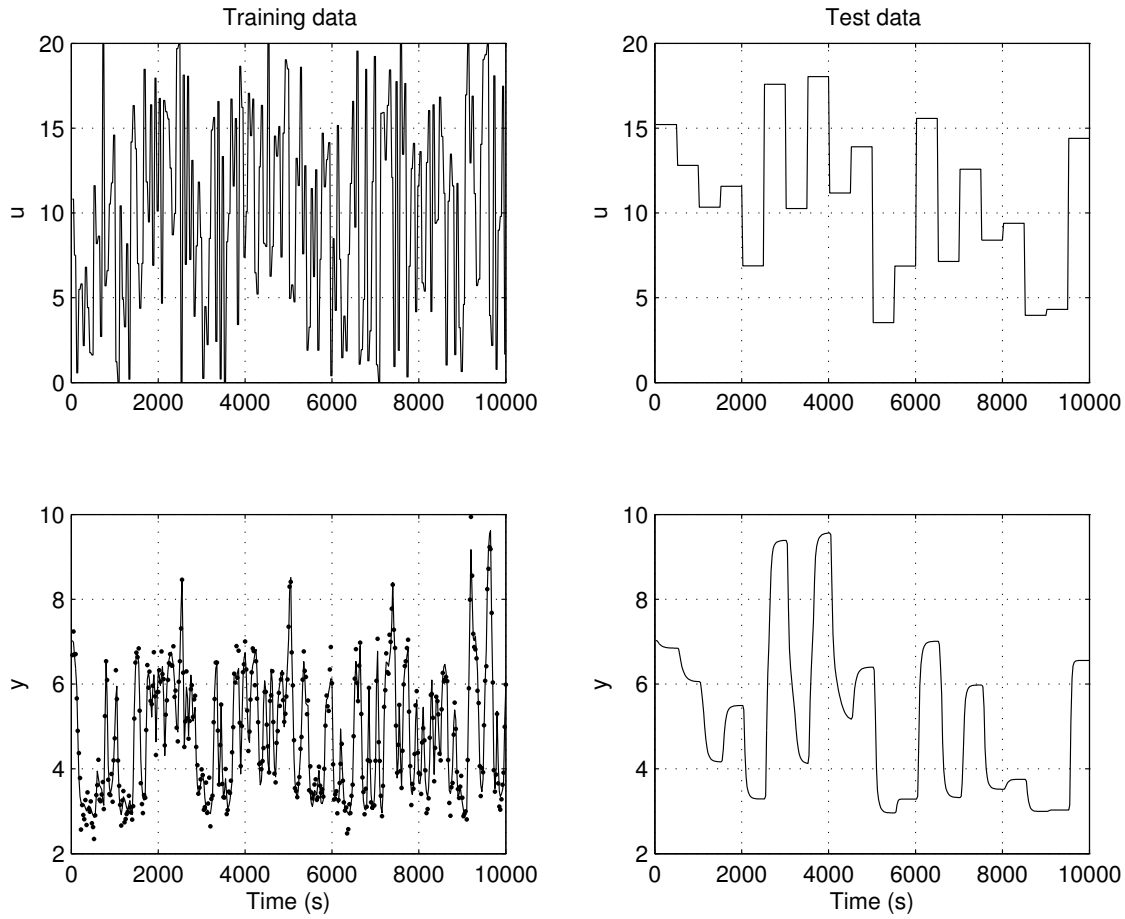


Figure 6.8: Input-output data for identification of the pH_B (and pH'_B) system; identification data (left) and validation data (right). The bottom left panel shows both the non-noisy data (solid line) and the data with 20% noise (dots).

6.1.5 Case Study: Steam Generator

This case study concerns the identification of a steam generating plant at the Abbott Power Plant in Champaign, Illinois, using input-output data obtained from the DaISy repository (De Moor, 2013). The unit is dual fuel (oil/gas) fired and performs both heating an electric power generation.

The aim is to identify a 4-input 4-output nonlinear plant model. A diagram of the process inputs and outputs is shown in Figure 6.9. Before identification is performed, the water level control is stabilized using feed-forward control and PID control (Espinosa & Vandewalle, 1999). The control signal for the feed-forward control is proportional to the steam flow. The PID controller is added to compensate the mass in the drum. The purpose of the identification is the specific control objective of preserving the level of the header pressure and the oxygen level in the flue gas (Pellegrinetti & Bentsman, 1996).

The four inputs are the fuel flow rate u_1, air flow rate u_2, water reference level u_3 and steam demand u_4 (disturbance defined by the load level). The four output variables are the steam pressure y_1, excess oxygen y_2, water level y_3 and steam flow rate y_4. The available data used for identification, shown in Figure 6.10, are obtained from the DaISy repository (De Moor, 2013). The total number of data points is 9600, out of which the first 7600 were used for training and 2000 points were used for testing. The sampling rate is 3

seconds. The modeling task is a challenging one: The plant exhibits high-order dynamics, and displays transportation delays due to the piping, which result in varying dead times, as well as large amounts of sensor noise.

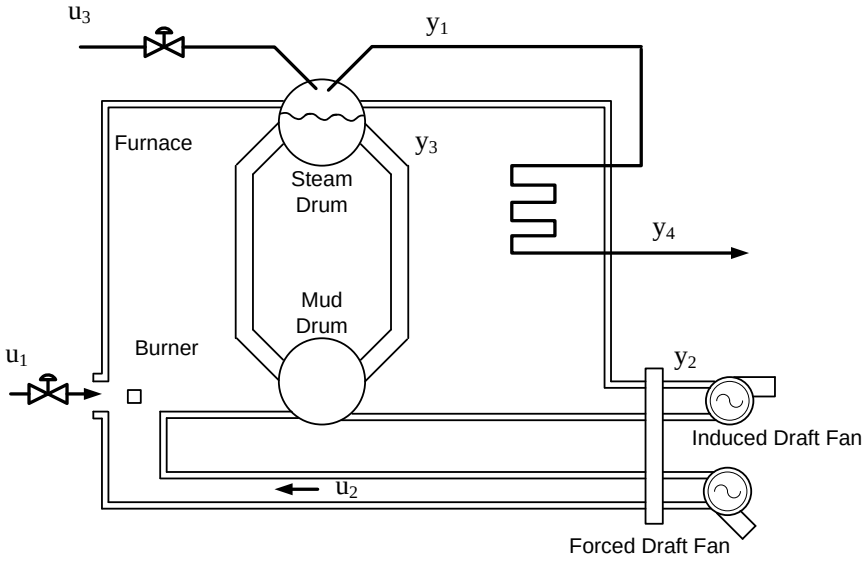


Figure 6.9: A diagram of the steam generator plant.

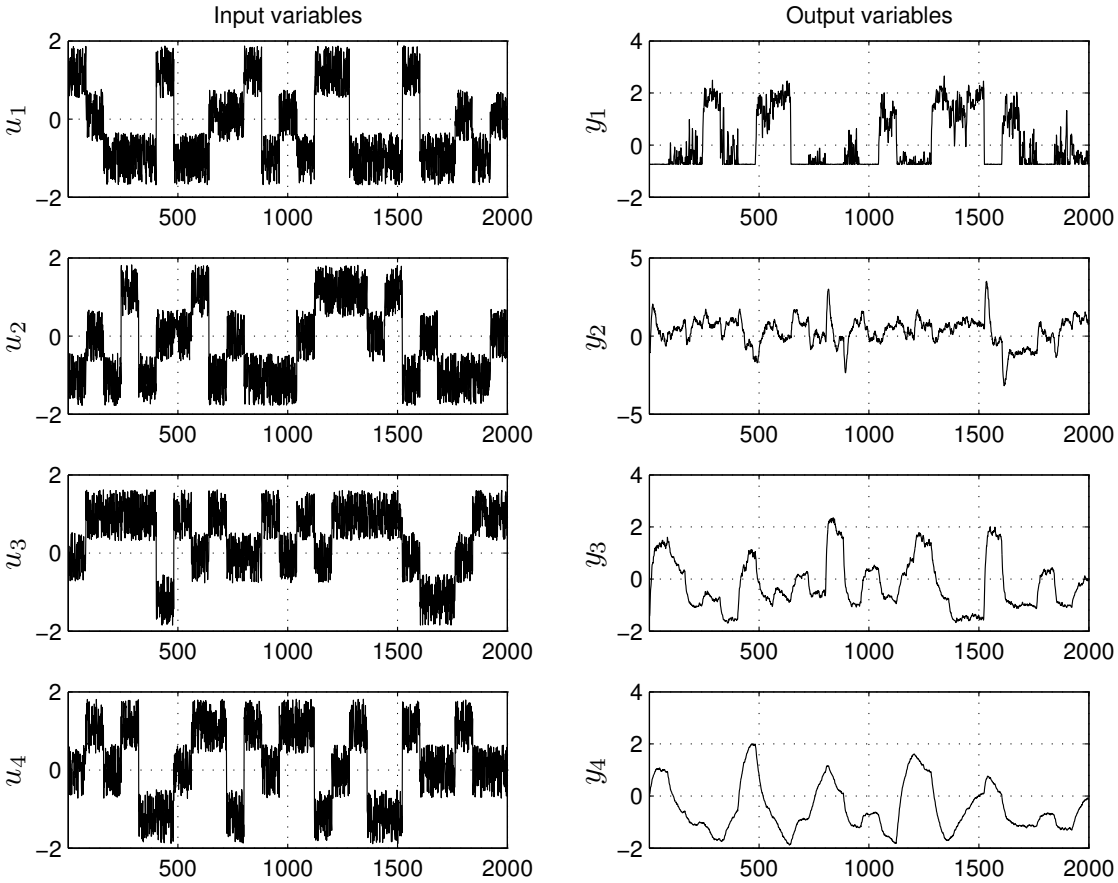


Figure 6.10: Input-output data of the steam generator dynamic system used for testing.

6.1.6 Case Study: Robot Arm

This case study deals with modeling a 7-degree-of-freedom anthropomorphic robot arm (Vijayakumar & Schaal, 2000), shown in Figure 6.11. The data comes from a robot arm which performs various rhythmic and discrete movement tasks. The robot arm system studied here is highly nonlinear and presents a modeling challenge.

The data is collected with a sampling rate of 0.1 s and consists of 21 input variables and 7 output variables. The 21 input dimensions are the 7 joint positions, 7 joint velocities, and 7 joint accelerations, while the 7 output dimensions are the 7 torque commands, $\tau_1, \tau_2, \cdots, \tau_7$, for each of the motors. The modeling goal is to approximate the torque commands of every robot motor in response to the vector of input variables.

The number of data points used in this study is 2000, where the first 1000 are used for training and the last 1000 are used for testing. One part of the training data is shown in Figure 6.12, where it is visible that the data is nonlinear and contains a small amount of noise.

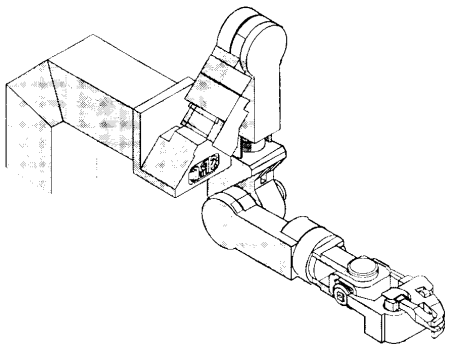


Figure 6.11: The 7-degree-of-freedom anthropomorphic robot arm.

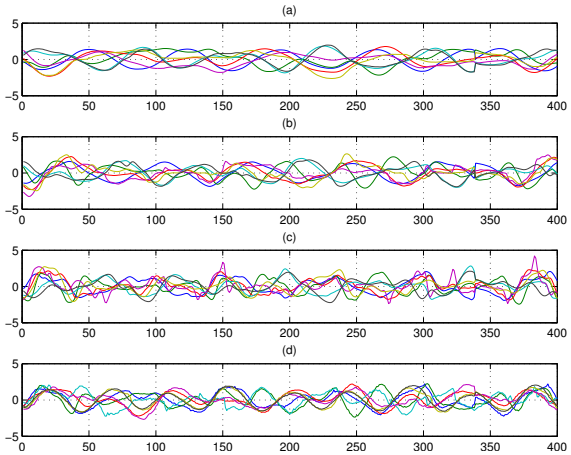


Figure 6.12: Input-output data for the robot case study, where only the first 400 normalized data points from the training set are shown. The figure depicts (a) the 7 joint positions, (b) the 7 joint velocities, (c) the 7 joint accelerations, and (d) the 7 torque commands (outputs).

6.1.7 Case Study: Winding Process

This process is a setup of an industrial winding process pilot plant (Bastogne, Garnier, & Sibille, 2001). It is composed of a plastic web and three reels coupled with direct-current motors. The web is unwinded from the first reel (denoted as unwinding reel), goes over the traction reel and is rewinded back on the rewinding reel, shown in Figure 6.13.

The task is to control the web tension in order to avoid sliding effects, wrinkles and material distortion. It presents a nonlinear and time-variant system. For this system, the three angular speeds S_1, S_2, S_3, are measured by using dynamo tachometers. Data about the setpoint current at the DC motor M_1, and the setpoint current at the DC motor M_2 is also available.

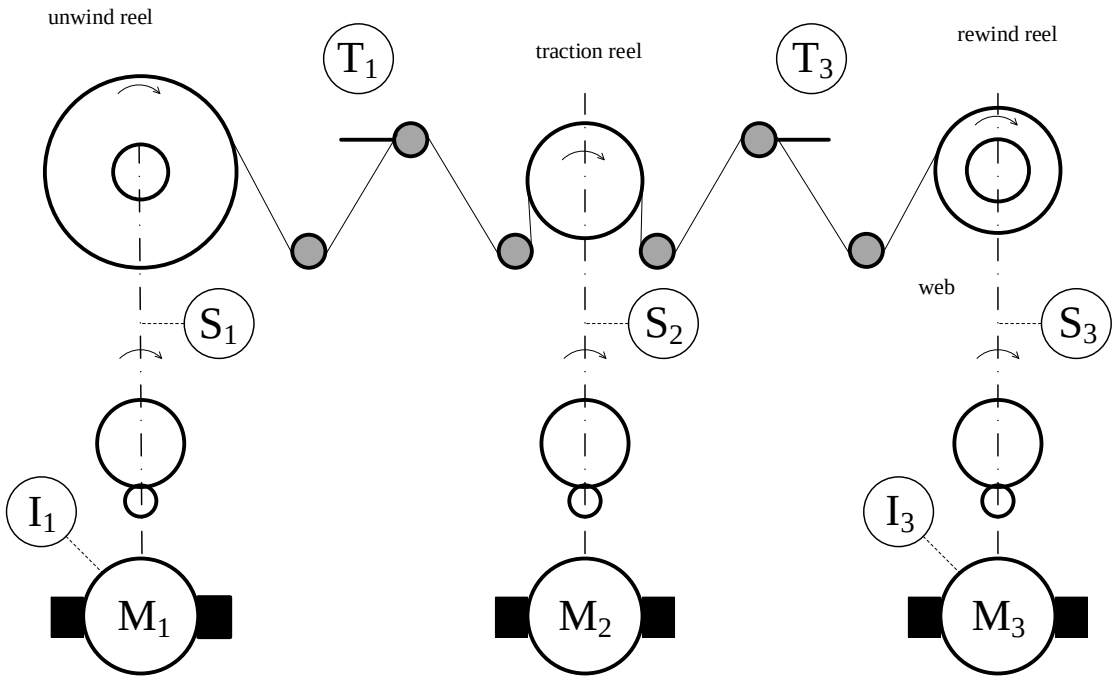


Figure 6.13: A diagram of the winding process.

The tensions in the web between the first and the second reel (T_1), and between the second and the third reel (T_2) are measured by tension meters. These two variables are the outputs, while the previous five are considered inputs (De Moor, 2013). The sampling rate is 0.1 s, while there are 2500 data samples available, covering a period of 250 s. The input-output data used are shown in Figure 6.14.

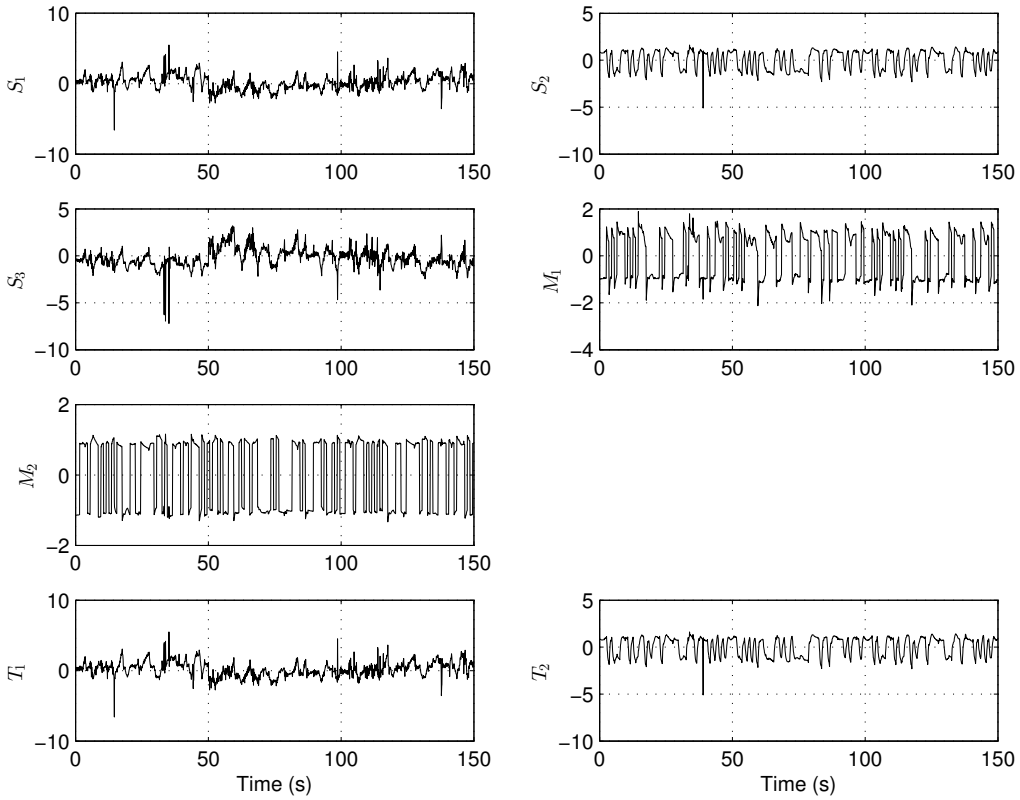


Figure 6.14: Input-output data for the winding case study. Only the training data are shown. The first three rows show the five input variables, while the last row shows the two output variables.

6.2 Datasets

This part first describes the preprocessing of the dynamic system measurements, which are performed for each of the dynamic system case studies, introduced earlier. It also describes the procedure for the determination of the optimal dynamic order. Finally, it summarizes the transformed datasets, which are used in the machine learning setting, for the evaluations that follow.

6.2.1 Preprocessing

The input-output data of the dynamic systems that are considered, require a preprocessing step. The transformation of the systems measurements is performed in order to successfully model the dynamic system. The measurements are transformed according to the external dynamics approach (Nelles, 1999). The approach is named "external" since there is a clear separation: external dynamics filter bank and a nonlinear static model (approximator) (Nelles, 2001). This approach requires that a value for the order (lag) is selected, which can be different for every variable. For example, the transformed data for a 1-input (q_c) 2-output (C_a and T), considering a lag of 2 for all variables, are illustrated in Figure 6.15.

time instant	*input vector* **x**						$f_2(\boldsymbol{x})$
3	$q_c(1)$	$q_c(2)$	$C_a(1)$	$C_a(2)$	$T(1)$	$T(2)$	$\hat{T}(3)$
4	$q_c(2)$	$q_c(3)$	$C_a(2)$	$C_a(3)$	$T(2)$	$\hat{T}(3)$	$\hat{T}(4)$
5	$q_c(3)$	$q_c(4)$	$C_a(3)$	$C_a(4)$	$\hat{T}(3)$	$\hat{T}(4)$	$\hat{T}(5)$
:	:	:					:
k-1	$q_c(k-3)$	$q_c(k-2)$	$C_a(k-3)$	$C_a(k-2)$	$\hat{T}(k-3)$	$\hat{T}(k-2)$	$\hat{T}(k-1)$
k	$q_c(k-2)$	$q_c(k-1)$	$C_a(k-2)$	$C_a(k-1)$	$\hat{T}(k-2)$	$\hat{T}(k-1)$	$\hat{T}(k)$

Figure 6.15: The transformed data using the external dynamics approach, and the simulation procedure are shown. The dynamic system has three variables q_c, C_a and T, and the chosen lag is 2. The first row shows the transformed data for time instant 3. The last column shows the predictions of the model for output variable T. The shaded cells in the subsequent rows, as well as the arrows, illustrate the simulation procedure.

In the system identification and control literature, it is common to test several lag combinations for a dynamic system under investigation. Furthermore, in some of the cases, different lag values are chosen for different variables, a choice which is influenced by the modeling technique selected. In this thesis however, we do not make a thorough investigation of the optimal lag for each variable of the system, and consider instead an identical lag for all variables.

The optimal value for the dynamic order (lag) for each case study was selected using a heuristic procedure. The training data for each of the case studies were further split into two subsets: a larger training subset and a smaller validation subset. Several different values for the order of the model were considered, as shown in Table 6.1. For each dynamic order, a multi-output model tree was built for the case studies that contained multiple output variables, and a single-output model tree for the others. The model tree was learned by using the training subset, and evaluated using the validation subset. The value for the order which gave rise to the smallest squared output error on the validation subset was selected. Also, all methods considered in the thesis, which also include Neural Networks and other neuro-fuzzy approaches, were evaluated using identical lag values. In other words, all of the methods were tested on the same static nonlinear regression problems. Table 6.1 also reports the number of features (regressors), when looking at the dynamic system identification problem as a static nonlinear function approximation.

Table 6.1: The dynamic system case studies considered, the selected lags and the dimensionality of the datasets obtained.

Case study	Orders considered	Num. inputs	Num. outputs	Order selected (SO)	Num. features; num. targets (SO)	Order selected (MO)	Num. features; num. targets (MO)
CSTR	1;2;3;4	1	2	3	9;1	3	9;2
GLS	1;2;3	2	2	1	4;1	2	8;2
Narendra	1;2	1	1	2	4;1	/	/
pH	1;2;3;4	1	1	2	4;1	/	/
Steam Gen.	4;5;6;7	4	4	6	48;1	6	48;4
Robot	1;2	21	7	1	28;1	1	28;7
Winding	1;2;3	5	2	3	21;1	3	21;2

6.2.2 Dataset Summary

The datasets obtained from the seven dynamic system case studies can be grouped in two groups, according to the number of output variables they contain. The two groups are:

- datasets for modeling single-output dynamic systems
- datasets for modeling multiple-output dynamic systems

The first group of datasets is shown in Table 6.2, while the second, multi-output group of datasets is shown in Table 6.3.

6.3 Selected Methods for Comparison

We compare the two model tree methods and ensembles thereof, with two methods that are well-established in the area of system identification: Neural Networks (Cristianini & Shawe-Taylor, 2000) and ANFIS (Jang et al., 1997). From the parameter identification perspective, both methods utilize global optimization of the parameters. This means that all the parameters are considered and optimized together in each optimization step (iteration). On the other hand, the model tree algorithms M5' and Lolimot use local optimization of the local model parameters, and local optimization of the structure parameters.

The models that the compared methods learn are of two types: a feed-forward neural-network model and a Takagi-Sugeno fuzzy model. The final models of Lolimot and the smoothed M5' tree are also equivalent to a Takagi-Sugeno mode, by their learning strategy is substantially different from the one employed by ANFIS. The latter uses a separate structure identification step and global optimization of the model parameters, while the model tree learning algorithm Lolimot uses an integrated structure identification and local parameter estimation approach. A brief overview of the properties of the methods is given in the following paragraphs.

We use feedforward Artificial Neural Networks (NN) (Cristianini & Shawe-Taylor, 2000), more specifically a multilayer perceptron with one hidden layer of neurons, trained by using a backpropagation procedure. The number of neurons in the hidden layer is the only parameter whose value needs selection. We use the Neural Network Toolbox implementation in Matlab. The network training is performed using the Levenberg-Marquardt optimization procedure.

Table 6.2: The generated datasets for the single-output machine learning analysis. The parenthesis, if present, denote the output/target variable.

	Case study	Output variable
1	CSTR(Ca)	concentration of A
2	CSTR(T)	temperature
3	CSTR'(Ca)	concentration of A (+20%n)
4	CSTR'(T)	temperature (+20%n)
5	GLS(p_1)	pressure
6	GLS(h_1)	level
7	Narendra	y
8	Narendra'	y (+20%n)
9	pH$_A$	pH
10	pH'$_A$	pH (+20%n)
11	pH$_B$	pH
12	pH'$_B$	pH (+20%n)
13	SteamGen(y_1)	drum pressure
14	SteamGen(y_2)	excess oxygen
15	SteamGen(y_3)	water level
16	SteamGen(y_4)	steam flow
17	Robot(τ_1)	torque comm. for motor #1
18	Robot(τ_2)	torque comm. for motor #2
19	Robot(τ_3)	torque comm. for motor #3
20	Robot(τ_4)	torque comm. for motor #4
21	Robot(τ_5)	torque comm. for motor #5
22	Robot(τ_6)	torque comm. for motor #6
23	Robot(τ_8)	torque comm. for motor #7
24	Winding(T_1)	tension btw. reel #1 and #2
25	Winding(T_2)	tension btw. reel #2 and #3

The Adaptive network based fuzzy inference system (ANFIS) (Jang et al., 1997) is a hybrid neural-network approach, which builds a Takagi-Sugeno fuzzy model. ANFIS solves the parameter estimation problem by using a hybrid learning rule that combines the backpropagation gradient descent and the least-squares estimation method. The structure identification task – determining the number of fuzzy rules and initial positioning of the fuzzy rule centers is a separate procedure that must be run before ANFIS. It can be approached using different methods: grid partitioning, fuzzy clustering of the instance space, or a tree-based approach (Jang, 1994). All of these can be used to determine the initial number and placement of the fuzzy rules. The first one produces an overly large set of rules for modeling problems with a large number of dimensions, and is only used for small problems. This leads to the known problem of the ANFIS method: it suffers from the curse of dimensionality as the number of input dimensions gets larger.

In this work, we use the Matlab implementation of the ANFIS method, which is available in the Fuzzy Logic Toolbox. For the structure identification problem, we utilize the computationally cheaper alternative, among the three alternatives mentioned before, which is the fuzzy c-means clustering method. We do not use the clustering method's automatic procedure of determining the number of clusters, however, since in our experience it produces sub-optimal models. Instead, we utilize a version of the clustering algorithm with a single tunable parameter: the number of fuzzy clusters.

Table 6.3: The datasets and the output variables considered in the multi-output machine learning analysis. The parenthesis denote the output/target variable for the multi-output case study.

	Case study	Output variable
1	CSTR(Ca)	concentration of A
2	CSTR (T)	temperature
3	CSTR'(Ca)	concentration of A (noisy dataset)
4	CSTR'(T)	temperature (noisy dataset)
5	GLS(p_1)	pressure
6	GLS(h_1)	level
7	SteamGen(y_1)	drum pressure
8	SteamGen(y_2)	excess oxygen
9	SteamGen(y_3)	water level
10	SteamGen(y_4)	steam flow
11	Robot(τ_1)	torque comm. for motor #1
12	Robot(τ_2)	torque comm. for motor #2
13	Robot(τ_3)	torque comm. for motor #3
14	Robot(τ_4)	torque comm. for motor #4
15	Robot(τ_5)	torque comm. for motor #5
16	Robot(τ_6)	torque comm. for motor #6
17	Robot(τ_8)	torque comm. for motor #7
18	Winding(T_1)	tension btw. reel #1 and #2
19	Winding(T_2)	tension btw. reel #2 and #3

6.4 Experimental Design

This part describes the experimental setup, the data used and the experimental procedure. For most of the dynamic system case studies, there is a data in the form of a training set and a test set already available. The training and test sets are obtained from different signals generated under the same conditions. This means that the input signals used to excite the dynamic systems (or their models) in both cases had the same form and similar extreme values.

However, the evaluation is performed on three separate sets of the data, instead of two. The available training set, for each of the case studies, is split using a 60:40 split into a training and validation part. This results in three separate sets of the data. The training and validation parts are used for obtaining the optimal parameter values for the considered methods, a procedure which is known as *early stopping*. After the optimal parameter values are determined, the final model is built on the whole training set, which includes both training and validation parts of the data. Finally, this model is evaluated on the unseen test data, and the results of this evaluation are reported.

To allow for a fair comparison, the three sets of data are exactly the same for all considered methods. As described, the first two are used for obtaining sensible parameter values for the (algorithm, dataset) combination. This parameter optimization step consisted of trying several discrete values for the parameters. We report the parameter values we tested for each of the methods, in Table 6.4.

The Lolimot model tree algorithm and its modifications are evaluated by running the algorithm for 30 iterations. In our experience, more than 30 iterations does not improve the performance of the Lolimot model tree. The Lolimot method includes an automatic

Table 6.4: Method parameters and the values considered in the experimental evaluation.

Method name	**Parameter name**	**Values considered**
Lolimot	Num. iterations	30
The modified Lolimot	Num. iterations	30
	k_σ	0.25 : 3.0
M5'	LS Regression: only M5' features	T/F
	LS Regression: M5' feature selection	T/F
	Fuzzification overlap	0.05:0.90
Bagging Lolimot and modified Lolimot	Number of trees	100
Bagging of M5' trees	Number of trees	100
Forest of M5' trees	Number of trees	100
	Size of random subset of feat. att	$[0.2; 0.4; 0.6; 0.8] \# features$
Neural Networks	Num. hidden neurons	1:15
ANFIS	Num. fuzzy clusters (rules)	2:8

determination of the optimal complexity, implemented by using the AIC (cf. Subsection 3.4.3). However, in our experience this measure does not yield the tree with the optimal size in all cases. Also, to make the comparison to the other methods fair, we employ the early stopping approach. The ensembles od Lolimot and the modified Lolimot, built by the bagging approach, are made up of 100 model trees. The bagging and (random) forests of M5' trees are also made up of 100 model trees.

To obtain more reliable estimates of the performance of the methods which include randomization, each experiment was repeated 5 times[1]. In each of repetitions we used different random seeds for the randomization procedure in the methods. In the experimental results, we report the mean and the standard deviation of the simulation (output), measured as RRMSE across the 5 runs.

The ensembles of model trees that we are proposing utilize a randomization procedure. The bagging and random forest methods use the randomization to select the bootstrap samples, to be used to learn each base model. Additionally, the random forest uses attribute randomization during the split selection procedure. The base learning algorithms, i.e., the model tree learning algorithms Lolimot and M5' do not use randomization. The other methods compared, ANFIS and Neural Networks both use randomization, however the randomization has a different influence on the model accuracies for the two methods. The Neural Networks use randomization to set the initial values of the neuron parameters. The structure determination of ANFIS, carried out by using c-means fuzzy clustering, uses random initial standard deviations of the fuzzy membership Gaussian functions. However, the randomization in ANFIS has very little effect on the predictive performance of the method, and this in turn might be considered as a positive property of the approach.

Multi-output experiments. This experimental analysis considers several multi-output case studies, which contain a total of 19 output variables, shown in Table 6.3. The results of the multi-output modeling are reported per output variable, considering and evaluating all 19 variables together. This is the approach used in the reporting of the number of wins, and in the statistical significance tests.

[1]From a machine learning viewpoint, a more reliable measure of the performance can be obtained by repeating the training and testing procedure several times and reporting the mean and variance of the error measure. We used 5 runs, to illustrate the influence of the random seed value chosen on the ensemble model.

6.4.1 Performance Measures

This subsection describes the evaluation of the performance of the models learned. The performance measures consider three aspects of the models: a) the predictive performance of the models, b) the time required for model learning and c) the size of the resulting model, in terms of the number of terminal nodes of the model tree, or the number of local models.

The predictive performance of the model determines how the model would perform on new unseen data points, or in other words, how well does the model generalize. The size of the model is reported in a) number of terminal nodes for individual model trees and b) average number of terminal nodes for the model trees in an ensemble.

The evaluation of the predictive performance of a dynamic system's model is carried out according to the purpose of the model and often requires a stringent and purpose-specific evaluation. When evaluating a model using one-step-ahead prediction, as shown in Figure 2.2 (a), the predicted values for the system variable are compared to the measured values. On the other hand, the procedure of simulation, illustrated in Figure 2.2 (b), introduces one substantial difference: the one-step-ahead model predictions are fed back to the model to produce predictions for the more distant future.

As discussed in 2.1.2, the first step of one-step-ahead prediction and simulation is the same. The simulation procedure, presents an iterative application of the predictive model. The result of the simulation is called simulation output and the corresponding error is denoted as simulation error or output error.

It is worth noting that the simulation procedure can also be seen as a form of generalization. The first data point in the training set corresponds to the first time step in the simulation procedure. Assuming that the model is not perfect and does not yield the exact output value of the training set in every subsequent time step, we can look at all the other data points as novel: the model has not been trained using them, and its performance on these novel data points might be considered as an estimate of its generalization ability.

Due to the realistic possibility of error accumulation in the case of an inaccurate model, divergence of the simulation predictions from the measured values may occur as we move further into the future. This increases the importance of simulation, as it represents a more stringent evaluation of the predictive performance of a dynamic systems' model.

The predictive performance of the obtained models in the empirical analysis is assessed by looking at a squared-error measure. Similar as in the previous chapter, we calculate and report the root relative mean-squared error (RRMSE), which is also known in the control domain as normalized root mean squared error (NRMSE):

$$RRMSE = \frac{\sqrt{\sum (y_i - \hat{y}_i)^2}}{\sqrt{\sum (y_i - \bar{y})^2}} \tag{6.11}$$

In a comparison of method A to method B, we report the number of wins, and the statistical significance of the difference for the measure. By a "win", we denote a strictly smaller value for the performance measure: smaller squared error, smaller model size, or a smaller amount of time required for learning. In the experimental analysis we report the number of wins the statistical significance of the difference. In some of the experiments, we report all three measures, while in others, where for example the comparison assumes models of the same size, we report only a subset of the three measures.

To test the difference in performance of algorithms on a number of datasets, we utilize the non-parametric Wilcoxon signed rank test. The p-value used for the Wilcoxon test is 0.01, or a 1% significance level. This means that when comparing methods A and B using some performance measure, we report the number of wins, discussed earlier, and the

p-value of the Wilcoxon test, denoted as w-test. The differences in predictive performance of the methods are also summarized by plotting the RRMSE values of the method A (x-axis) against the corresponding errors of method B (y-axis). In the case the markers are grouped and close to the diagonal, the methods perform similar. In case the markers are above the diagonal, method A shows better predictive performance than method B, and vice versa.

Besides the predictive performance, this chapter also depicts the auto-correlation of the output error. The analysis of auto-correlation properties of the error could potentially lead to conclusions regarding the bias of the parameter estimates in the models. There is a possibility that the bias in the parameter estimates goes undetected if only the prediction/output errors are analyzed.

6.5 Evaluating Modifications of the Model Tree Learning Algorithms

This section evaluates the proposed modifications to the two model tree learning algorithms. It considers the relatively accurate fuzzy model tree algorithm Lolimot, built for the purpose of system identification and the relatively fast and general-purpose model tree algorithm M5'. As outlined in the previous sections, the main differences are a) the former builds soft trees, while the latter build crisp ones, b) the split selection: the former is a look-ahead type model tree algorithm, while the latter is a fast heuristic model tree algorithm and c) the parameter estimation.

This part is organized as follows. First, we consider the M5' method. We compare its performance to Lolimot, analyze M5's performance and introduce modifications. Then, we consider the Lolimot method and evaluate its modifications. Finally, we summarize the results obtained, and try to answer the question: What are the benefits of using general-purpose model tree algorithms, which use crisp model trees and crisp local model estimation, for modeling dynamic systems.

In more detail, the M5' modifications concern the post-smoothing of the local models. The modifications are performed after the M5' tree is learned and are performed to obtain a more powerful model - soft tree. The soft model tree would potentially lower the overall model error, and fix the discontinuities on boundaries between local models.

Also, the modifications of Lolimot analyzed here are several. The first one is aimed at a more efficient evaluation of candidate splits. It evaluates whether replacing some of its expensive calculations with potentially faster procedures would produce a method which runs faster and has similar accuracy. The next two modifications can be summarized as a modified search for the optimal tree structure, in terms of splits and MSF overlaps. After this, we analyze whether it is beneficial to use the simulation procedure in the split evaluation heuristic. At the end, we evaluate a different approach for LM estimation, by using the global parameter estimation procedure.

6.5.1 Evaluating M5' Modifications

6.5.1.1 Comparing M5' to Lolimot

This part aims to put the two model tree algorithms in context and provide a fair comparison of the performance of M5' and Lolimot. As the pruning procedure of M5' could not be directly tuned to obtain the tree with desired complexity, the following strategy is used: The M5' tree is built, and its number of local models is taken as a reference. A Lolimot tree of the same size, denoted by Lolimot$_S$, is built and compared to the M5' algorithm.

The summarized results of the output error are shown in Figure 6.16, while the results of the statistical tests are shown in Table 6.5. They show that Lolimot$_S$ with the same number of local models outperforms the M5' algorithm. The difference is statistically significant at the 1% level. The complete results, which also include the number of local models, and the one-step-ahead error, are shown in Table A.11.

Table 6.5: A statistical comparison of M5' to Lolimot$_S$. A summary of Table A.11.

M5' : Lolimot$_S$	RRMSE	Time
#wins	3:22	18:7
w-test	0.0	0.0

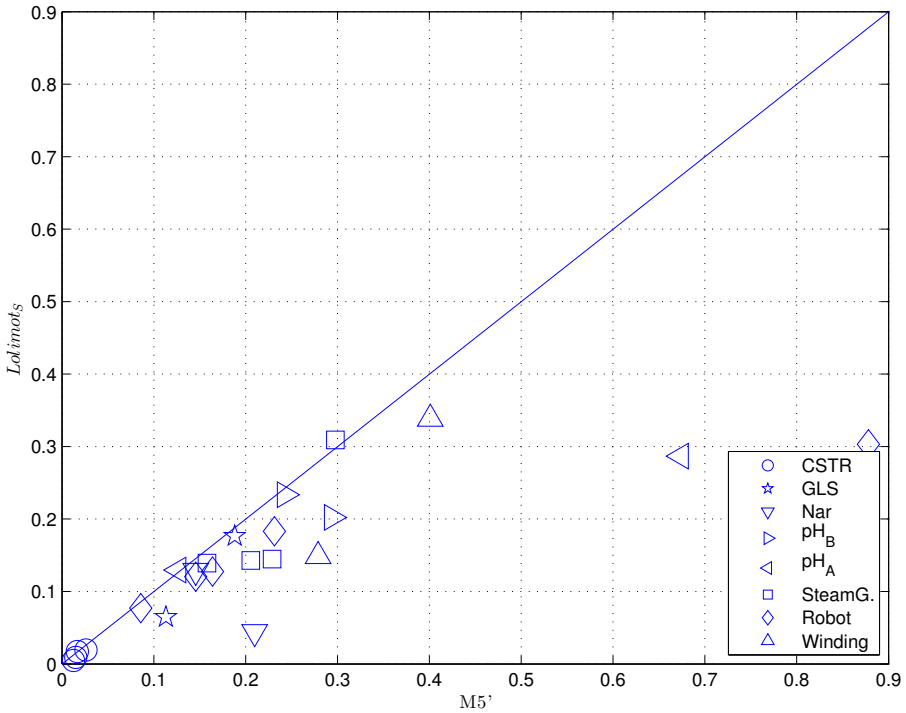


Figure 6.16: A comparison of the predictive performance of crisp M5' model trees to soft Lolimot$_S$ model trees of the same size.

To summarize, the comparison of the performance of the equally-sized M5' and Lolimot model trees (M5' vs Lolimot$_S$), shows that the Lolimot$_S$ builds trees which perform better or equal to M5'. By analyzing the results, we can also conclude that:

- Lolimot shows better performance than M5', with the exception of only a few cases with similar performance
- For several datasets the M5' pruning mechanism overprunes, i.e., leaves only a few local models; Such are for example almost all datasets for the Steam Generator case study
- For some of the robot datasets, the Lolimot model tree performs substantially better

These results suggest that the size of the M5' tree that the method has determined during pruning is not always optimal and the method could benefit from a modification of the

post-pruning phase. In the case of considering a method which would select the splits by means of a look-ahead procedure, the post-pruning phase might not be necessary.

6.5.1.2 Replacing the Crisp Local Model Estimation with Fuzzy

This experiment tries to evaluate the benefits of a soft model tree, as opposed to a crisp one. It compares the M5' crisp model tree and its performance, to a soft model tree, derived from the crisp one. The soft tree contains fuzzy splits, which are implemented with MSFs exactly like in Lolimot. Also, the local models are estimating by solving the weighted least-squares regression problem, again using the same procedure as in Lolimot.

The procedure in this experiment is: First, the tree structure is learned by M5'. Then, the tree is converted to a soft tree and the local models are estimated. Both steps are performed identically as in Lolimot. The resulting model, denoted as $M5'_{SOFT}$, is evaluated and its performance is reported.

The results in Figure 6.17 show that the replacement of the crisp model tree with a soft/fuzzy one increases the performance of the model tree. The errors are mainly above the diagonal, which indicates that the $M5'_{SOFT}$ variant provides an increase in the performance.

Table 6.6: A statistical comparison of the output error of $M5'_{SOFT}$ to M5' and Lolimot$_S$. A summary of Table A.12.

$M5'_{SOFT}$:	M5'	Lolimot$_S$
#wins	16:8	3:21
w-test	0.018	0.002

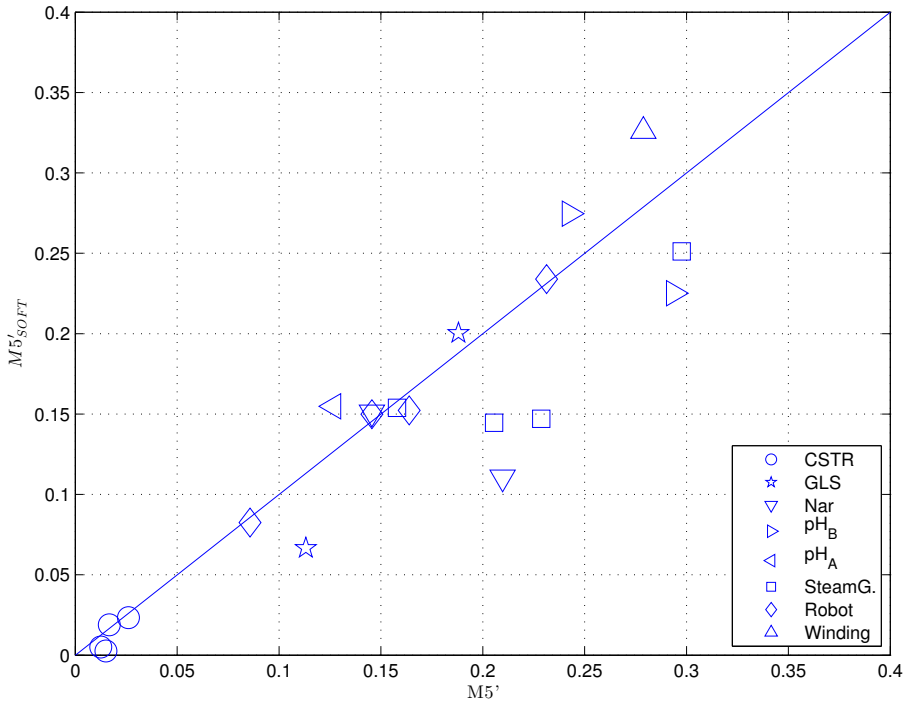


Figure 6.17: Replacing the crisp local model estimation with fuzzy. A comparison of the output error of $M5'_{SOFT}$ and M5'.

Regarding the statistical comparison of the results, shown in Table 6.6, which also provide a comparison with Lolimot$_S$, we can conclude that:

- $M5'_{SOFT}$ performs better than M5', however the difference is not statistically significant at the 1% level
- $M5'_{SOFT}$ performs worse than Lolimot$_S$ and the difference is statistically significant.

The complete results, which also include the complexity of the models, are available in Table A.12.

6.5.1.3 Evaluating Smoothing Variants

This part would try to assess whether a smoothing procedure performed after the tree structure is learned and parameters of linear models estimated is beneficial. The aim here is to evaluate two alternatives for smoothing of the local model predictions. The two alternatives are the built-in M5' smoothing, denoted as $M5'_{BSM}$, and smoothing by fuzzyfication, denoted as $M5'_{Fuzz}$, both introduced in Subsection 4.1.1. As discussed there, the smoothing by fuzzification could be taken into account for the linear model estimation. However, given that the linear model parameters in M5' are estimated on an unpruned and potentially very large tree, this would lead to a large amount of calculations required.

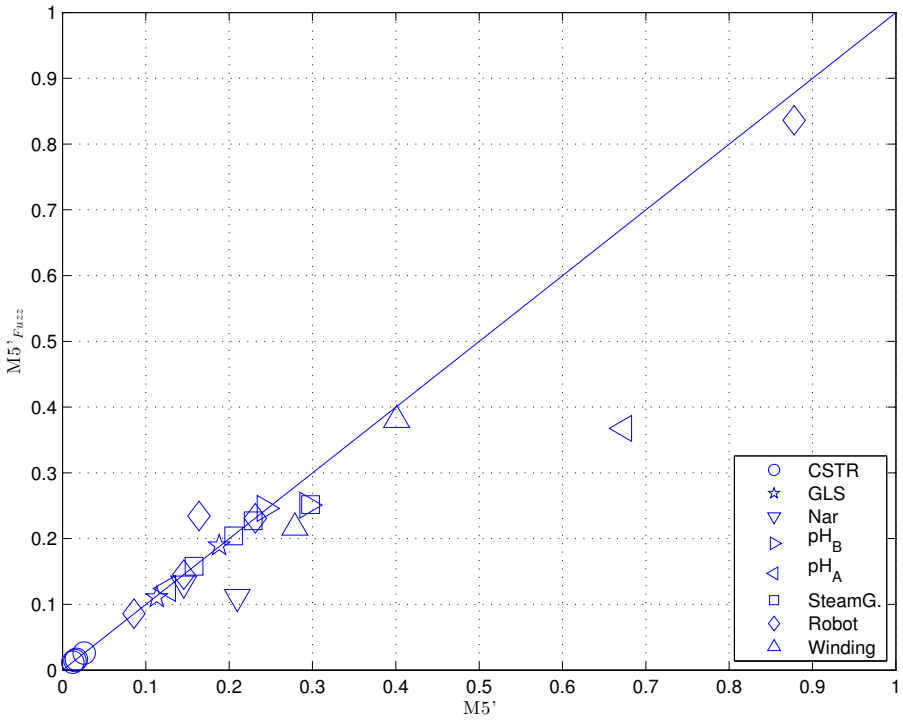


Figure 6.18: Testing the effectiveness of M5' without smoothing, as opposed to $M5'_{Fuzz}$.

What follows are summarized results, whether the two types of smoothing of the M5' tree offer a difference in the performance, as compared to the case of no smoothing. The results in Table 6.7 show how do the two smoothing alternatives compare to the unsmoothed model trees, in terms of output error and time required for learning. The $M5'_{BSM}$ variant wins in 22 out of the 25 cases, while the $M5'_{Fuzz}$ wins in 18 cases, where the differences are statistically significant at the 1% level only for the first case. This means that both types of smoothing increase the performance over the unsmoothed M5' model tree.

Additionally, the output error results of M5' and $M5'_{Fuzz}$ are depicted in Figure 6.18. It shows a large increase in performance for the Narendra and pH case studies. The complete numerical results, along with the time required to build each model, can be found in Table A.13.

Table 6.7: A statistical comparison of the effectiveness of M5' smoothing. In both cases the tree sizes are equal. A summary of Table A.13.

M5' :	M5'$_{BSM}$ RRMSE	M5'$_{Fuzz}$	M5'$_{BSM}$ time	M5'$_{Fuzz}$
#wins	3:22	7:18	19:6	25:0
w-test	0.001	0.045	0.704	0.000

6.5.2 Evaluating Lolimot Modifications

The modifications of the Lolimot algorithm that are evaluated in this subsection consider the evaluation of the candidate splits, the search for the optimal tree structure, the utilization of the output error while learning and the estimation of local model parameters with a different approach.

In the first part, we analyze the potential benefit of a more efficient candidate split evaluation. The next two parts consider the modified search for the optimal tree structure. It generates and evaluates more than one candidate split per dimension, which results in different positioning of the MSF centers. In other words, it tries to determine if better positioning of the MSF can influence the final model performance. Recall that, the Lolimot algorithm builds two equal hyper-rectangles and places the MSFs in their centers. Our modification builds two hyper-rectangles of different sizes, and places the MSFs in their centers. Also, the modifications related to the tree structure consider different overlaps of the MSFs. However, the modification preserves Lolimot's procedure of determination of the MSF deviation as a function of the partition size. With our proposed modification, the MSF deviations are calculated using a different function of the partition size.

Another modification is aimed at the estimation of the local model parameters. The two variants evaluated are local and global estimation of parameters. The fuzzy procedure of local parameter estimation is faster, but it does not offer the best predictive performance. The alternative global parameter estimation is considerably slower, however it may offer the best predictive performance as it also takes into account the MSF overlaps. Finally, this part would analyze the differences for modeling multi-output dynamic systems.

6.5.2.1 Modified Evaluation of Candidate Splits

Here we asses the modification to the heuristic evaluation of candidate splits. We report the output error performance of the obtained models and time needed for model building. We denote the two variants with Lolimot and Lolimot$_{ME}$. Both variants utilize the same values for the tree complexity. This value is determined by using the early stopping procedure, i.e., with the help of the validation set. The k_{SIM} value for the Lolimot$_{ME}$ variant is chosen by using a trial-and-error approach, and is set to 350. This means that we evaluate the candidate splits only for 350 steps.

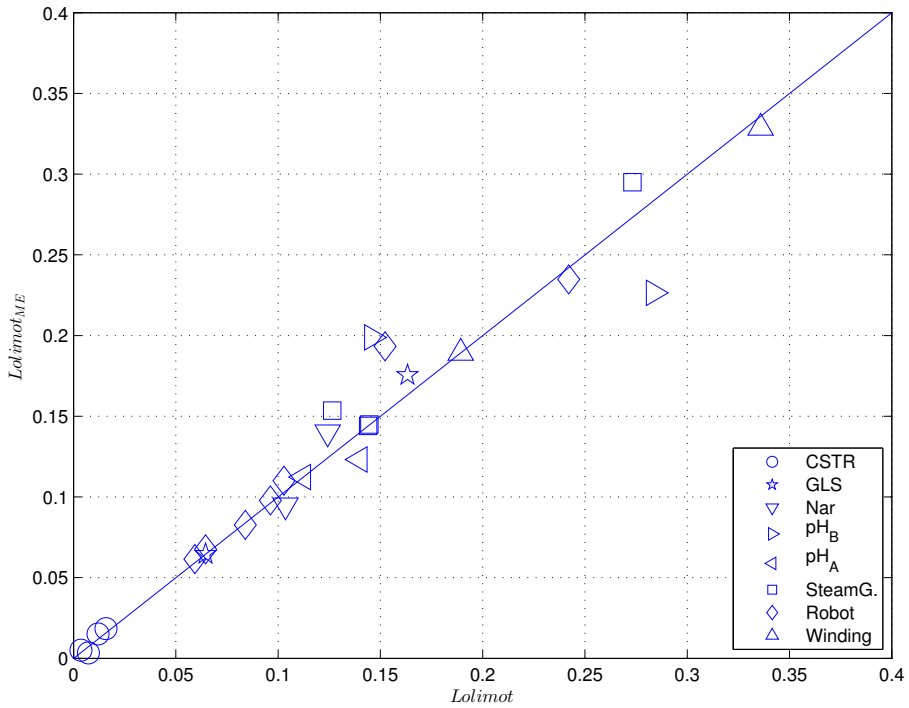


Figure 6.19: A comparison of the output error of Lolimot$_{ME}$ and Lolimot.

The output error results are shown in Figure 6.19, while the results of the statistical tests for the errors and building time are shown in Table 6.8. The complete results are available in the Appendix A, Table A.14.

Table 6.8: A statistical comparison of Lolimot and Lolimot$_{ME}$. A summary of Table A.14.

Lolimot$_{ME}$: Lolimot	RRMSE	Time
#wins	7:18	24:1
w-test	0.274	0.0

The output error results show no statistically significant difference between the original and modified split evaluation procedure. Also, the figure shows that the errors are close to the diagonal of the plot, which indicates similar performance. The time needed to build the models is on the other hand quite different for the two variants. The modified split evaluation procedure shows improvement of up to several times (see Table A.14). The differences in the times needed for model building are statistically significant. We can conclude that the Lolimot$_{ME}$ variant is a better choice, since it offers similar performance as the original method, but with reduced learning times. In the analysis that follows, we will utilize the Lolimot$_{ME}$ variant.

6.5.2.2 Modified Search for an Optimal Tree Structure

This part is going to evaluate two modifications of Lolimot. The first one considers *several split cut-points* and the second one utilizes *different MSF overlap* values.

The first modification evaluates whether considering more than one half-split can result in increased performance. In this analysis, we employ versions of the algorithm which evaluate 2, 4, and 8 splits in each dimension instead of only one. The modified versions

of the algorithm are named Lolimot$_{C2}$, Lolimot$_{C4}$, and Lolimot$_{C8}$, respectively. All three versions are built upon the Lolimot$_{ME}$ variant, evaluated in the previous part.

Table 6.9 presents the results from the statistical tests, which indicate that there is no statistically significant difference in the simulation performance. On the other hand, the three algorithm versions need more time for learning and build larger trees. The results shown in Figure 6.20 confirm that the performance of Lolimot$_{C8}$ is similar or worse to the original algorithm, as the points are around and above the diagonal. The complete results are shown in Tables A.15 and A.16.

Table 6.9: A statistical comparison of Lolimot$_{ME}$ with Lolimot$_{C2}$, Lolimot$_{C4}$, and Lolimot$_{C8}$. A summary of Table A.15 and Table A.16.

	RRMSE			Time			Size		
Lolimot$_{ME}$:	Lol$_{c2}$	Lol$_{c4}$	Lol$_{c8}$	Lol$_{c2}$	Lol$_{c4}$	Lol$_{c8}$	Lol$_{c2}$	Lol$_{c4}$	Lol$_{c8}$
#wins	14:11	13:12	16:9	25:0	25:0	25:0	11:14	7:18	9:16
w-test	0.427	0.786	0.090	0.000	0.000	0.000	0.400	0.588	0.781

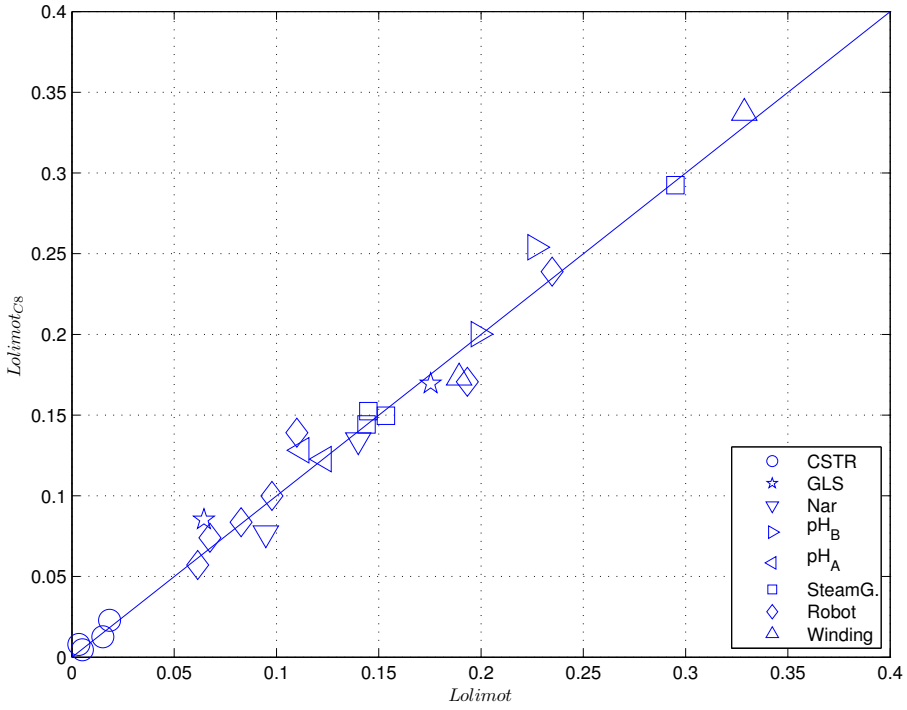


Figure 6.20: A comparison of the performance when considering several cut-points. Lolimot$_{ME}$ and Lolimot$_{C8}$ are shown.

The second modification evaluates different values for the k_σ parameter (cf. Eq. (4.10)), which effects the amount of MSF overlap. The values considered for k_σ are the following: $4, 2, \frac{1}{0.75}, 1, \frac{1}{1.25}, \frac{1}{1.5}, \frac{1}{1.75}, \frac{1}{2.0}, \frac{1}{2.25}, \frac{1}{2.5}, \frac{1}{2.75}, \frac{1}{3.0}, \frac{1}{3.5}$. The modification evaluated in this part is denoted as Lolimot$_{ksig}$. Again, it is built upon the the Lolimot$_{ME}$ variant, i.e., it both uses the more efficient split evaluation and it optimizes the k_σ parameter.

Table 6.10: A statistical comparison of Lolimot$_{ME}$ and Lolimot$_{ksig}$. A summary of Table A.17.

Lolimot$_{ksig}$: Lolimot$_{ME}$	RRMSE	time	size
#wins	19:6	4:21	4:21
w-test	0.024	0.0	0.007

The results in Figure 6.21 show that the errors are mostly below and around the diagonal of the plot, which means that there is some difference in performance visible. This result is not statistically significant at the 1% level, however, the p-value is quite small, and we believe that the optimization of the overlap has the potential to improve the result. In the investigations that follow, we will be using Lolimot$_{ksig}$.

Regarding the running times and sizes of the trees, Lolimot$_{ksig}$ builds trees with more local models, as compared to the other variant, and this also requires more time. The differences in model size and learning times are statistically significant.

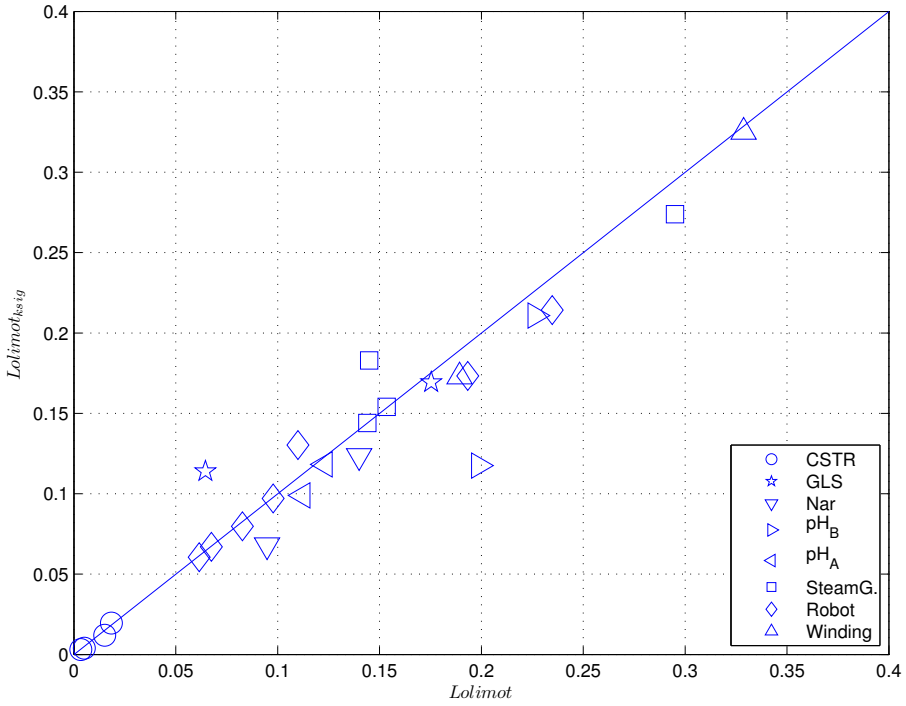


Figure 6.21: A comparison of the performance when optimizing the fuzzy MSF overlap .

As a summary of the search for optimal tree structure, we can conclude that the more important of the two sub-tasks is the optimization of the overlaps. The two experiments showed that only the overlap tuning shows improvement in the performance over the original method. The consideration of several cut-points for the splits does not show improved performance. It may be necessary to tune all MSF-related parameters, i.e., both the split cut-points and the overlaps in a single tuning procedure. However, this poses a more complex optimization problem, which also adds to the computational time required for modeling. Due to this result we presume that the dominating factor regarding the tree structure optimization in soft model trees is the amount of overlap, possibly coupled with the type of local model estimation.

It is worth noting that other fuzzy modeling methods like ANFIS optimize both the

position and the overlap of each fuzzy MSF. In the tree setting, this would translate to simultaneous tuning of both the split cut-points and the fuzzy MSF overlaps.

6.5.2.3 Utilization of the Output Error While Learning

This part analyzes whether the evaluation of candidate splits with simulation has any advantages over the evaluation with prediction. The Lolimot variant using the prediction error for evaluation of the candidate splits is denoted as Lolimot$_{osa}$. It is built upon the Lolimot variant which uses the more efficient split evaluation procedure, and the k_σ tuning. This means that this evaluation also tunes the values of k_σ, as well as the complexity of the tree. Both are performed by using the validation set. For the experiments reported here, both variants Lolimot$_{osa}$ and Lolimot$_{ksig}$ use same size of the tree and k_σ value.

The results of the comparisons are available in Table 6.11, where Lolimot$_{osa}$ shows worse performance in 16 out of the 25 tests. However this difference is not statistically significant. It is interesting to note that Figure 6.22 shows that there is considerable benefit of using the simulation error for the noisy pH variant, the measured GLS level variable and one of the winding variables. Given that the first is a noisy variant and the second is a measured variable, where the measurements are not ideal for identification, one might conclude that for practical applications, it is better to use the simulation error. There is no benefit in using the prediction errors, as the running times are different, but the differences are very small (cf. Table A.18).

Table 6.11: A statistical comparison of the Lolimot models built by using the output or the prediction errors. A summary of Table A.18.

Lolimot$_{osa}$: Lolimot$_{ksig}$	RRMSE	time
#wins	9:16	9:16
w-test	0.145	0.019

The results show that the evaluation of the candidate splits using simulation is better, when the intended use of the model is simulation. This is a result that might be considered as expected. This comparison was also performed in the work of Aleksovski, Kocijan, and Džeroski (2014a), where the authors conclude that the evaluation using the output error can bring slight improvement to the performance of multi-output model trees and ensembles, based on the Lolimot algorithm.

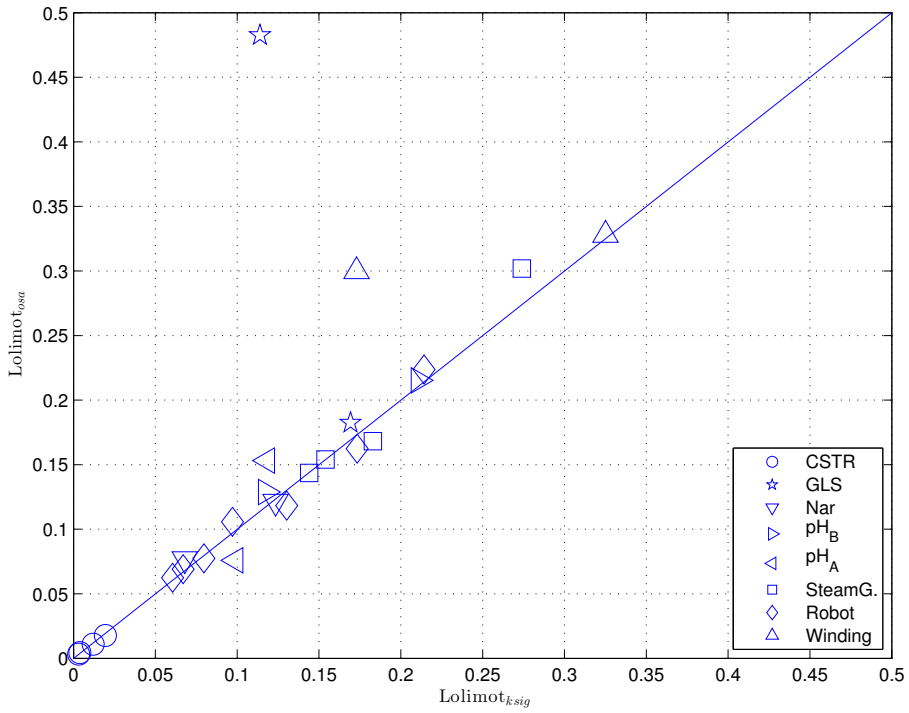


Figure 6.22: A comparison of the Lolimot models built by using the output or the prediction errors.

6.5.2.4 Global Parameter Estimation

This part compares the Lolimot method to a modified version where the local estimation is replaced by global parameter estimation. The modified version is denoted as Lol$_{GPE}$. This evaluation is reported differently than the previous ones, and only illustrates the potential that the global parameter estimation has. The reason is that from a machine learning point of view, the determination of the optimal complexity using the training and/or validation sets fails in most of the cases to find the optimal complexity. In the following we will present and discuss the results, and at the end we will outline some possible solutions for the complexity determination issue.

Figure 6.23 shows the comparison of local to global estimation in Lolimot. The plot shows the performance of the models for the single-output systems, in each of the iterations of the algorithm. It can be concluded that the global estimation in most cases obtains a model with smaller error, and it does this by using a smaller number of local models. After this optimal point is reached, the performance of the model tree deteriorates very fast. For example, Lolimot$_{GPE}$ produces results that outperform Lolimot for all the CSTR alternatives, the GLS dataset, the Narendra dataset and other. On the other hand, there exist datasets like the Steam-generator variants, for which the local estimation performs better.

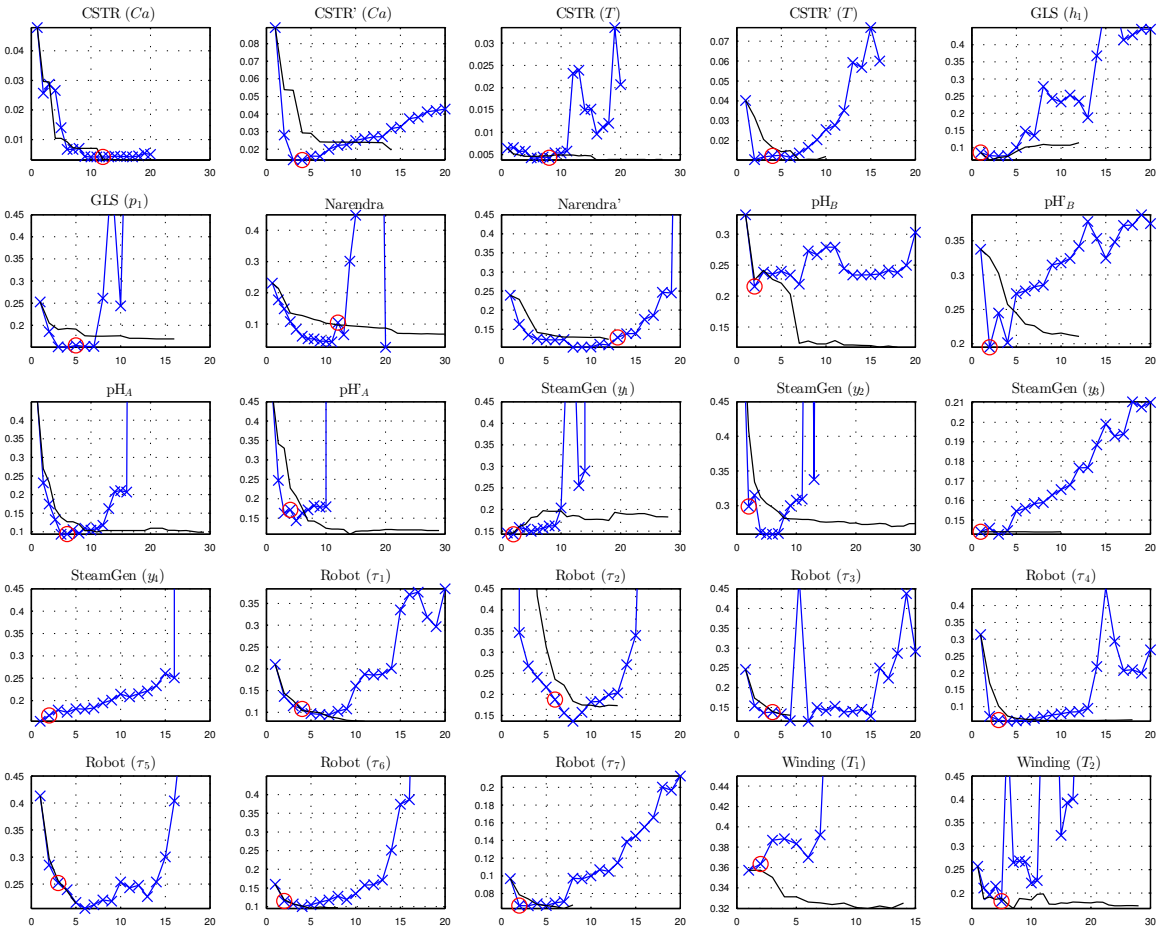


Figure 6.23: Performance of Lolimot (solid line) and Lolimot$_{GPE}$ (solid line with crosses) on the single-output datasets. Performance on the testing sets (unseen data) shown. For the latter, the complexity determined by the validation set is circled.

Note that in the figure, the selected model complexity is shown as a) a stop in the dashed line for the local estimation and b) a circle for Lolimot$_{GPE}$. Both of these are determined by using early stopping, i.e., with the validation set. It can be seen that this selected number of LMs for Lolimot$_{GPE}$ is in most cases suboptimal. So, the determination of the optimal complexity of the Lolimot$_{GPE}$ model presents a challenge. However, the obvious advantages of global estimation are that it can produce models with much lower output error, and in most cases, it performs this with a lower number of local models.

Regarding the potential benefits of local over global estimation, the work of Nelles (2001) has already outlined several results and conclusions for the Lolimot method. The main advantage of the local parameter estimation is the reduction of the effective number of parameters, and the good resilience to noise, which follows from the former property. However, the global parameter estimation is not considered for Lolimot. The advantage of the global parameter estimation is the potential to obtain a model with increased accuracy, as compared to the local model estimation. Its disadvantage is its computational complexity: the comput.complexity is $O(m^2)$, for m local models.

From the aspect of machine learning, we might conclude that the utilization of global parameter estimation in soft model trees requires a modified tuning procedure for the optimal model complexity, as the early stopping procedure fails. Instead, one might resort to techniques such as the AIC or BIC measures.

To summarize, this part presented some potentially interesting and novel results. They are:

- Lolimot$_{GPE}$ can achieve better performance with less local models, i.e., smaller complexity,
- Lolimot$_{GPE}$ is very sensitive to the number of local models and the optimal complexity is a challenge to determine.

For the remainder of the experimental analysis, the modified Lolimot method with the following two modifications:

- modified evaluation of the candidate splits,
- optimizing the MSF overlaps, by tuning k_σ,

would also be used and evaluated. This version of the Lolimot method would be denoted as L++.

6.5.2.5 Evaluating Multi-target Model Trees

This experiment evaluates the multi-target Lolimot and the modified version L++ for modeling multi-output dynamic systems. Its aim is to assess whether differences exist between the two methods, when building multi-target model trees. The results of the comparison on the 19 output variables are summarized in Figure 6.24 and in Table 6.12.

Table 6.12: A statistical comparison of the Lolimot and L++ models for multi-output modeling. A summary of Table A.19.

L++$_{MO}$: Lolimot$_{MO}$	RRMSE	time	numLM
#wins	11:8	11:8	9:10
w-test	0.421	0.017	0.255

It is visible in Figure 6.24 that the L++ method shows decreased output errors for all output variables for the CSTR and five of the seven variables for the Robot multi-output system. This is partially influenced by the different number of LMs selected by the early stopping procedure in the two runs. Also, the L++ method shows similar performance for both outputs of the GLS system, and slightly worse performance for both outputs of the Winding system. However, as Table A.19 shows, the running time of L++$_{MO}$ is smaller, by a factor of three to fifteen times. Exceptions are the cases where the number of iterations for the two methods is not comparable.

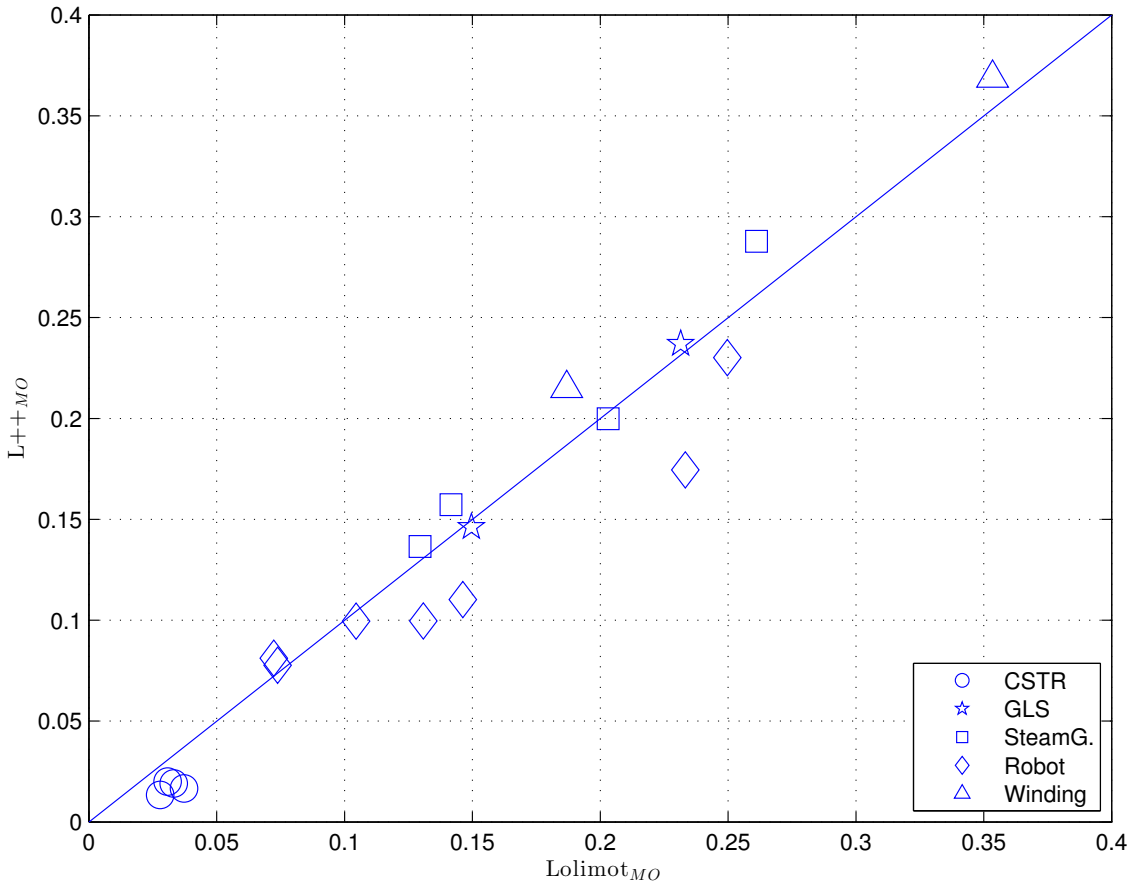


Figure 6.24: Comparison of the predictive performance of Lolimot$_{MO}$ and L++$_{MO}$. Each marker represents the performance of the methods on one output variable. The marker shape determines which dataset the variable belongs to.

6.5.3 Summary

In summary, the results showed that the soft or fuzzy estimation of local model parameters, and a soft model tree formalism are more flexible and produce more accurate models. For example, the experiment reported in Subsection 6.5.1.2 showed that the predictive performance of the crisp M5' model tree could be increased by re-estimating the local models with soft estimation and converting to the soft model tree formalism.

In Subsection 6.5.1.3 the variants for post-smoothing were empirically evaluated. Both variants resulted in increased performance of the model tree, however, we consider that the post-smoothing does not solve the whole problem. The incorrect determination of the tree structure, in terms of split attributes and cut-points, and also the incorrect determination of the linear model coefficients, present a bigger problem, and in many cases this could not be fixed by the post-smoothing. Examples of the incorrect determination are the models with large output error for the datasets Robot τ_2, τ_4, τ_7, pH'$_A$, Narendra and Narendra', shown in Figure 6.16 and Table A.11.

The possibility of using a general-purpose look-ahead tree learning approach, which builds crisp trees can be considered not flexible, when compared to soft tree approaches. It lacks the flexibility that the soft tree approaches have regarding: a) estimation of the local model coefficients and b) the tuning of the split cut-point. In more detail, a crisp model tree approach needs to determine if the amount of data in a partition is enough

for estimating a local model, or there is enough data, but some of them are colinear (e.g., input signal which does not change its value for several time steps). This is required for estimation of the correct local model parameters. The soft model tree approaches, on the other hand utilize weighted least-squares regression, which uses all data points in the local model estimation. This solves the mentioned issues with the local model parameters. Additionally, the crisp model tree approaches using look-ahead need to fine-tune the split cut-point, and this presents a computationally expensive procedure. As a comparison, the performance of a soft model tree approach such as Lolimot was not influenced by the tuning of the split cut-point, and the half-splits produced satisfactory results, as Subsection 6.5.2.2 showed. In our opinion, the overlap of the MSFs and the regression which uses all data help to decrease the influence of the cut-point optimization issue.

However, the fuzzy parameter estimation and soft model trees have the following disadvantages:

- the need to optimize the MSF parameters: MSF positions and overlaps,
- the computational cost of calculating the fuzzy validity functions for each data point, which may be problematic for a large number of local models and many repetitions (considering many candidate splits).

The modifications to the Lolimot method, summarized as L++, showed a decrease of running times, of up to several times (for some examples two times faster, for others up to 20 times). The optimization of the overlap parameter proved to be useful, and in many cases increase the performance of the model tree.

The analysis of using prediction or output error while learning showed that the output error proves more useful in the noisy cases and in the cases where the measurements are not perfect. The analysis also showed that there is no benefit of using the prediction error instead of the output error, as the running times are practically identical.

Also, the utilization of the global parameter estimation, as compared to the local estimation used in Lolimot, showed that the prediction results can be further improved. The issue there was that the optimal model complexity was hard to determine, and a model tree with only one or two LMs more than that optimum already had quite a deteriorated performance.

6.6 Model Trees and Ensembles for Single-output Modeling

This section reports the analysis of the ensembles of model trees. It considers bagging ensembles of Lolimot and L++, as well as bagging and random forests of M5' model trees. So far, the single trees built by the original and improved Lolimot methods showed better predictive performance than the ones built by M5'. However, we expect a potential improvement in the performance of the model when M5' is used in the ensemble setting. This is the reason why we include results using ensembles of M5' model trees.

The analysis treats both the single-output and multi-output problems. The number of trees parameter for the ensembles analyzed here was set to 100. A previous work (Aleksovski et al., 2014a) considered bagging of Lolimot model trees with different sizes, and concluded that acceptable performance may be achieved by using 50 trees, or less. In this thesis, however, we utilize a larger number of trees.

6.6.1 Lolimot vs Ensembles

This part analyzes whether the ensembles of Lolimot trees offer a difference in the performance, as compared to the case of a single Lolimot tree. The summarized results are

available in Figure 6.25, while the complete numerical results can be found in Table A.20.

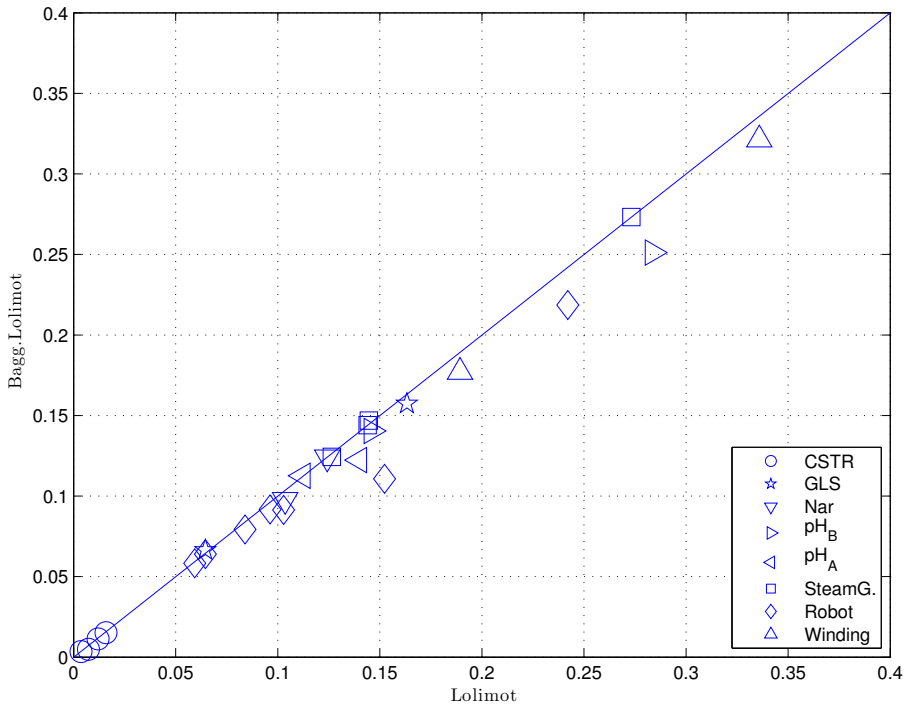


Figure 6.25: Evaluating the single-output predictive performance of Lolimot trees and Bagging of Lolimot trees.

The results suggest that bagging improves the performance over a single Lolimot tree, as most of the points are below the diagonal. The difference is statistically significant, as shown in Table 6.13.

Table 6.13: A statistical comparison of a single Lolimot tree and a bagging of Lolimot trees. A summary of Table A.20.

Lolimot :	Bagging RRMSE	time
#wins	5:20	25:0
w-test	0.0	0.0

6.6.2 Modified Lolimot vs Ensembles

This part presents the summarized results, which compare the modified Lolimot method (L++) to Bagging of L++ model trees. The experiment considers only the single-output problems. The results are summarized in Table 6.14 and Figure 6.26, while complete results are available in Table A.21.

Table 6.14: A statistical comparison of L++ and Bagging of L++. A summary of Table A.21.

L++ :	Bagg.L++	
	RRMSE	time
#wins	6:19	25:0
w-test	0.040	0.0

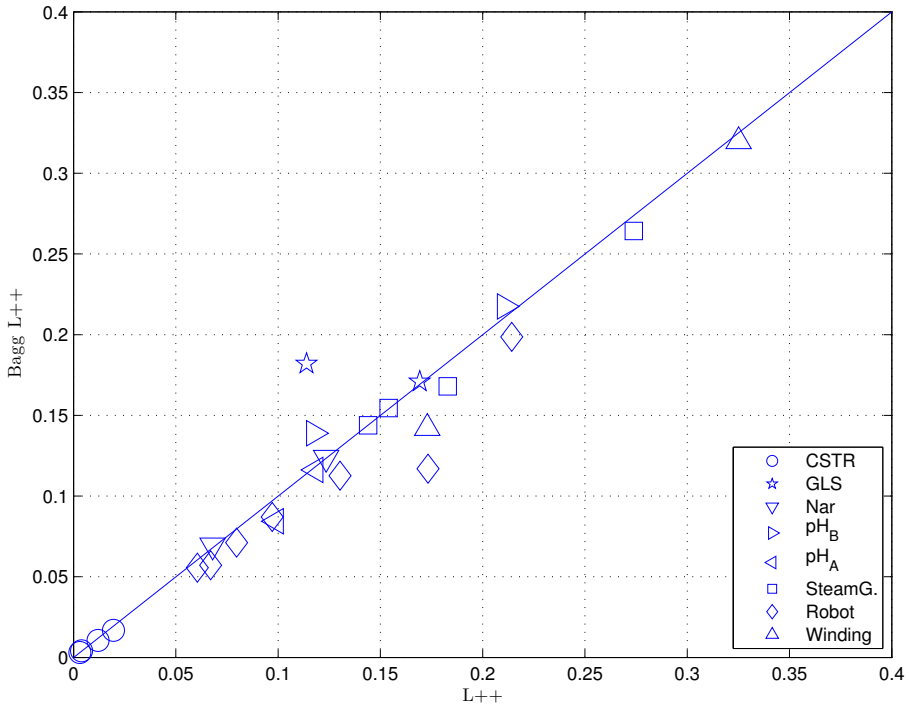


Figure 6.26: Evaluating the single-output predictive performance of L++ trees and Bagging of L++ trees.

The results show that the bagging procedure produces a smaller output error in 19 of the 25 cases, however, the difference is not statistically significant at the 1 % level. Figure 6.26 shows that the bagging procedure improves the performance for both Winding outputs, for most of the Robot outputs and most of the Steam Gen. output variables. Additionally, it showed that the bagging increases the output error for the two GLS output variables and the two pH_B variants.

6.6.3 Model Tree Ensembles vs Neural Networks and ANFIS

This part presents the comparison results of ensembles of both types of model trees, and selected methods frequently used for dynamic system identification. The latter include Neural Networks and the hybrid ANFIS method, introduced and discussed in Section 6.3. The summarized results are divided in two parts: the first compares the ensembles of L++ and M5' model trees, while the second part compares the ensembles of L++ to NNs and ANFIS.

Table 6.15 summarizes the results of the comparison between the different types of model tree ensembles. Due to the reasons discussed earlier, we also evaluate ensembles

of M5' model trees. The ensembles of M5' evaluated are bagging of M5' model trees (BMT) and forests of M5' model trees (FMT). Both of these include the ensemble selection procedure, proposed to increase the performance of ensembles of M5' trees, as discussed in Section 4.3.2.

Table 6.15: A statistical comparison of ensembles of the two model tree types. A partial summary of Table A.22 and Table A.23.

Bagg.L++ :	BMT RRMSE	FMT	BMT Time	FMT	BMT Complexity	FMT
#wins	19:6	19:6	1:24	1:24	4:21	6:19
w-test	0.006	0.009	0.0	0.0	0.003	0.008

The results in Table 6.15 show that the ensembles of L++ model trees build more accurate models than both types of ensembles of M5' trees. The difference is statistically significant at the 1% level for all paired output error comparisons. However, the ensembles of M5' require less time for learning the models, and also tend to build model trees with a smaller number of terminal nodes. This may be explained by the possible overpruning, i.e., leaving only a few terminal nodes in the model trees. This was discussed in Subsection 6.5.1.1, where it was identified as one of the reasons for inaccurate models.

Next we consider the comparison of the more successful among the ensemble approaches, the Bagging of L++ model trees, to Neural Networks and ANFIS. The results summarized in Table 6.16 show that Bagging of L++ MTs produces models with smaller output errors, as compared to Neural Networks and ANFIS. The differences in predictive performance are statistically significant at the 1% level.

Table 6.16: A statistical comparison of ensembles of model trees to NNs and ANFIS. A partial summary of Table A.22 and Table A.23.

Bagg.L++ :	NN RRMSE	ANFIS	NN Time	ANFIS	NN Complexity	ANFIS
#wins	19:6	18:7	0:25	7:18	4:21	3:22
w-test	0.003	0.008	0.0	0.109	0.0	0.0

However, the model trees included in the ensembles have larger complexity than the other methods. The Bagg.L++ require more time to build than NNs, while the time comparison to ANFIS does not show statistically a significant difference. It is worth noting that the reason that a competitive method like ANFIS provided results with worse predictive performance than Bagg.L++ is most likely the suboptimal parameters chosen by using the validation set. The rest of the results for the comparisons, including one-step-ahead results, are available in Appendix A, Table A.22.

6.6.4 Auto-correlation of the Output Error

This subsection depicts the auto-correlation of the output error of the tree-based models. First, it shows the auto-correlation of Lolimot and ensembles of Lolimot model trees. Then, it considers the M5' model trees and ensembles thereof.

Figure 6.27 presents the auto-correlation of the output error for Lolimot and ensembles of Lolimot, while Figure 6.28 presents the same for M5' and ensembles thereof. It is worth noting, that the simulation procedure for the single M5' tree learned on the Robot(τ_2) dataset diverges. Additionally, an M5' ensemble substantially improves the predictive performance over a single M5' tree for the Robot(τ_4) dataset, which may be related to the smaller auto-correlation of the ensemble error.

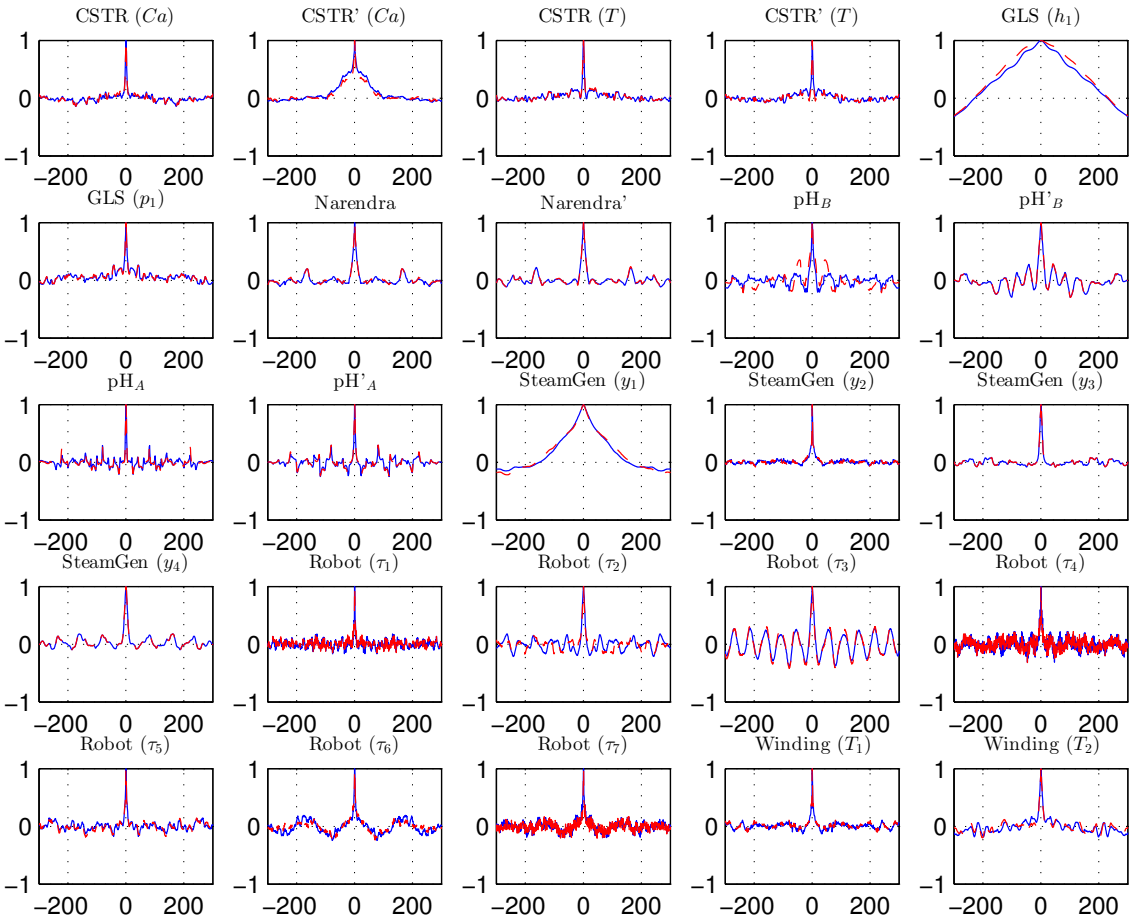


Figure 6.27: Auto-correlation of the output error of Lolimot (solid line) and ensemble of Lolimot (dashed line), on the single-output datasets. The x-axis denotes the lag.

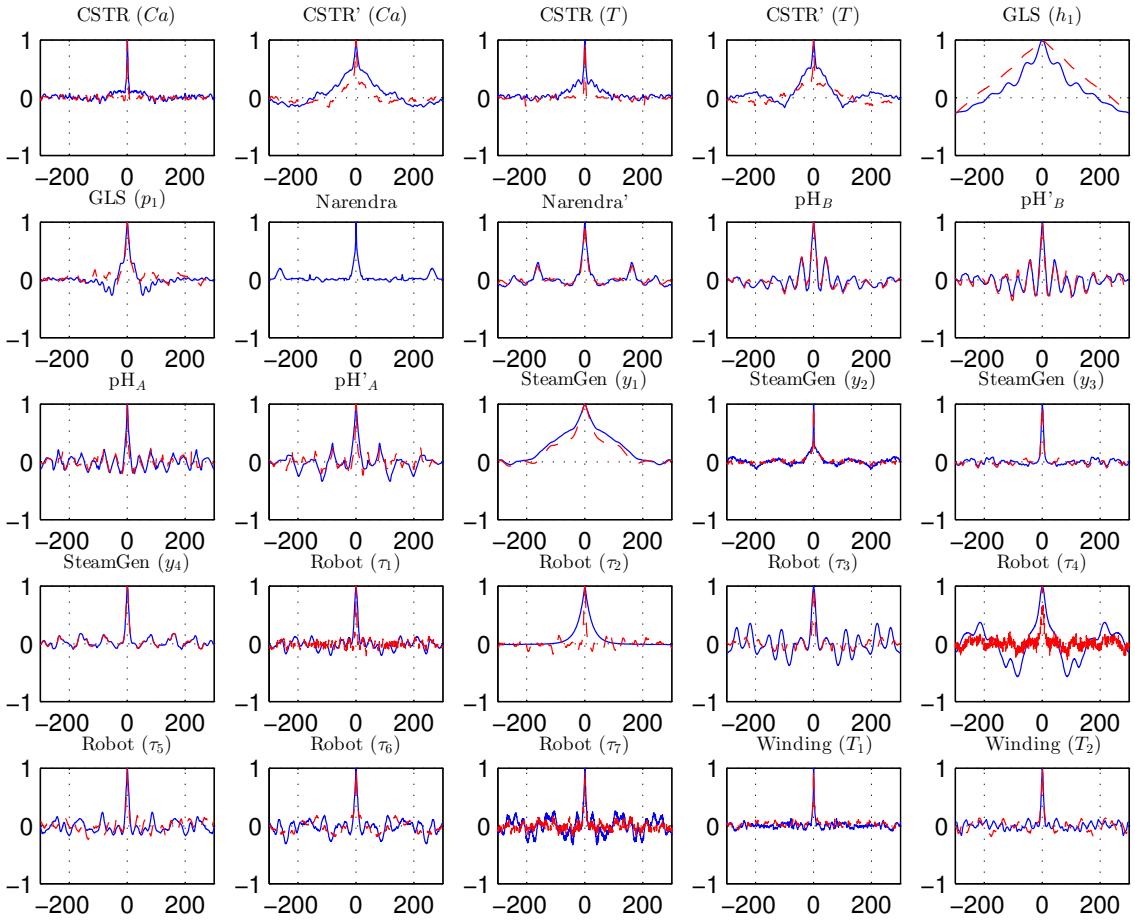


Figure 6.28: Auto-correlation of the output error of M5' (solid line) and ensemble of M5' (dashed line), on the single-output datasets. The x-axis denotes the lag.

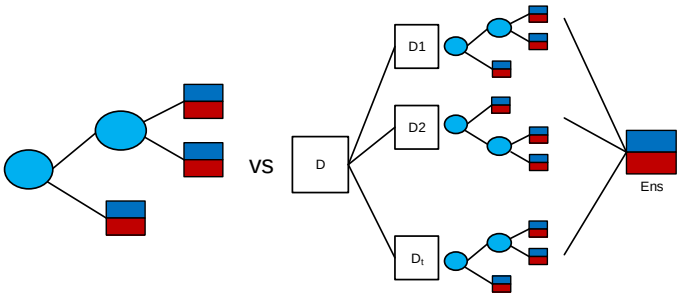
6.7 Model Trees and Ensembles for Multiple-output Modeling

This part evaluates the model tree and ensemble approaches for modeling multi-output systems. In particular, it empirically compares:

- a single multi-output model tree to an ensemble of multi-output model trees,
- building several single-output models, where each model predicts one output variable, to a multi-output model, where all output variables are predicted simultaneously.

For easier understanding of what is being compared, each of the subsections includes a graphic, representing the comparison on a sample 2-output system. The model trees which are able to predict multiple outputs are denoted by two different colors in their terminal nodes.

6.7.1 Modified Lolimot vs Ensembles



This part reports the results of the comparison of the multi-output version of L++ to bagging of multi-output L++ models. The results are depicted in Figure 6.29, and suggest that the bagging of multi-output L++ model trees improves the performance. The most of the markers are below the diagonal, while the others are very close to it, which shows the improved predictive performance in favor of bagging $L++_{MO}$. The improvements are visible for all outputs of the GLS and Robot case studies. On the other hand, there is no improvement in performance for the output variables of the Steam Gen. and CSTR case studies. The overall difference in predictive performance is statistically significant, at the 1% level, which is shown in Table 6.17. The complete results are available in Table A.24.

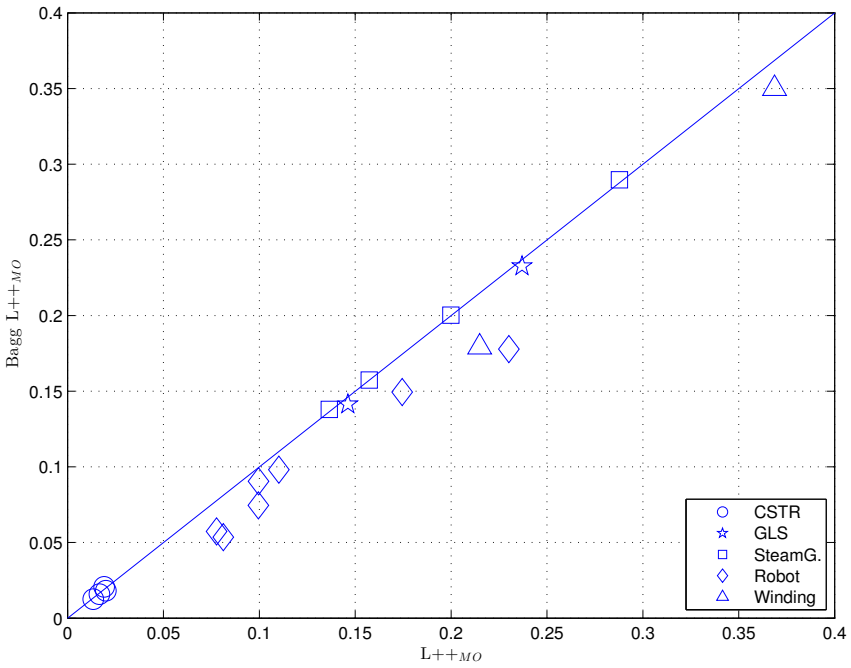
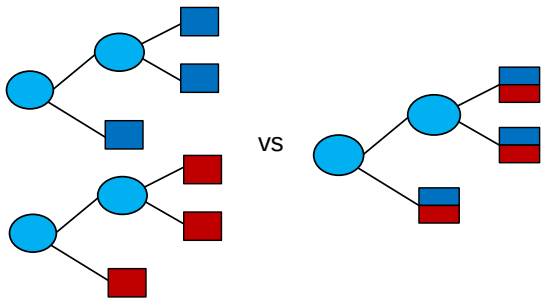


Figure 6.29: A comparison of the predictive performance of multi-output model trees to ensembles of multi-output model trees. Results for each of the output variables are shown separately.

Table 6.17: A statistical comparison of multi-output model trees to ensembles of multi-output model trees. A summary of Table A.24.

Bagg.L++$_{MO}$:	L++$_{MO}$ RRMSE	time
#wins	14:5	0:19
w-test	0.003	0.000

6.7.2 Several Single-output Models vs One Multi-output



In the case of **several single-output model trees, compared to a one multi-output model tree**, the results shown in Figure 6.30 suggest improvement in the predictive performance can be expected by using a multiple-output model tree. For example, the figure shows that all output variables of the Steam Gen. case study and five variables of the Robot case study, the multi-output alternative offers an improvement in performance. Additionally, the figure does not show markers for the noisy CSTR dataset (CSTR') and for the GLS dataset. The reason is that the single-output models diverged when simulated using parallel simulation. Table 6.18 also shows that the difference in predictive performance is statistically significant.

The complete results, available in Table A.25, also show that the total complexity of the model trees is smaller in the multi-output case. For example, the multi-output tree for the Steam Gen. contains only 21 LMs, while the four single-output model trees contain a total of 69 LMs. Also, the total time needed to build the four single-output model trees is larger than the one single-output model.

Table 6.18: A statistical comparison of separate single-output model trees, each predicting one output variable, to one multi-output model tree. A summary of Table A.25.

(Separate L++ SO models) : (L++$_{MO}$)	RRMSE	Time	Num.LMs
#wins	6:13	4:15	0:19
w-test	0.006	0.021	0.000

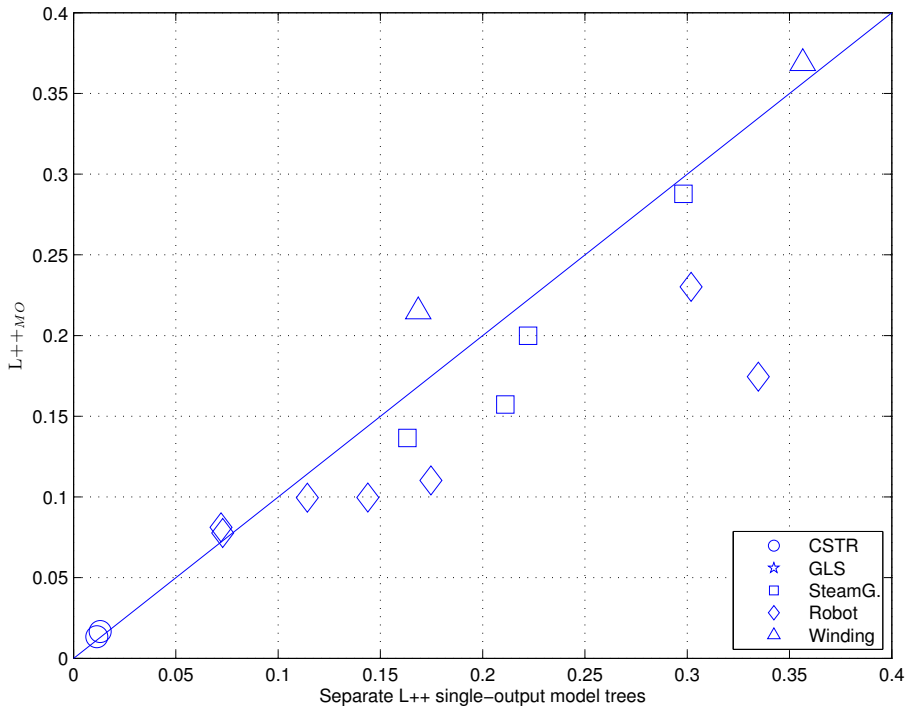
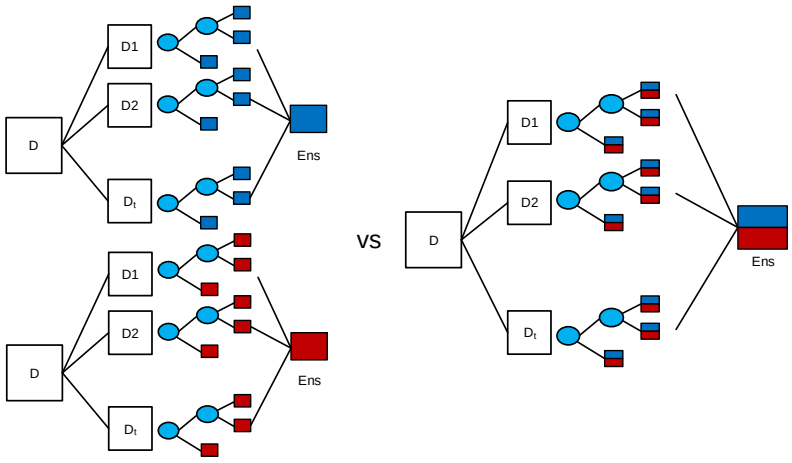


Figure 6.30: The predictive performance of several single-output L++ model trees, and the predictive performance of a multi-output L++ model tree.



Also, we analyze the case of **several ensembles of single-output model trees, compared to one ensemble of multi-output model trees**. The analysis is performed by using bagging ensembles of the L++ model trees. The results of this comparison are depicted in Figure 6.31, where it can be seen that the predictive performance is improved for all output variables of the Robot dynamic system, and three of the four outputs of the Steam Gen. system. Similar to the previous experiment, this figure does not show markers for the noisy CSTR dataset (CSTR') and for the GLS dataset, due to the divergence of the single-output models. It is worth noting that these two present realistic modeling problems as the first includes 20% noise, and the measurements of the second are not ideal. From this aspect, we can conclude that for multi-output problems it is better to resort to the multi-output Lolimot and L++ (and their ensembles), then the single-output versions.

Table 6.19 shows the results of the statistical tests, which suggest that the differences

in predictive performance are statistically significant at the 1% level. The results of the running times are also in favor of the multi-output modeling approach, and this difference is also statistically significant. The complete results, which include the one-step-ahead errors, learning times and complexities, are available in Table A.26.

Table 6.19: A statistical comparison of separate bagging of single-output L++ model trees, one for each output, to a bagging model which utilizes multi-output L++ model trees. A summary of Table A.26.

(Separate ensemble of SO L++ MTs) :	(Single ensemble of MO L++ MTs)	RRMSE	Time	Num.LMs
	#wins	5:14	4:15	0:19
	w-test	0.007	0.001	0.000

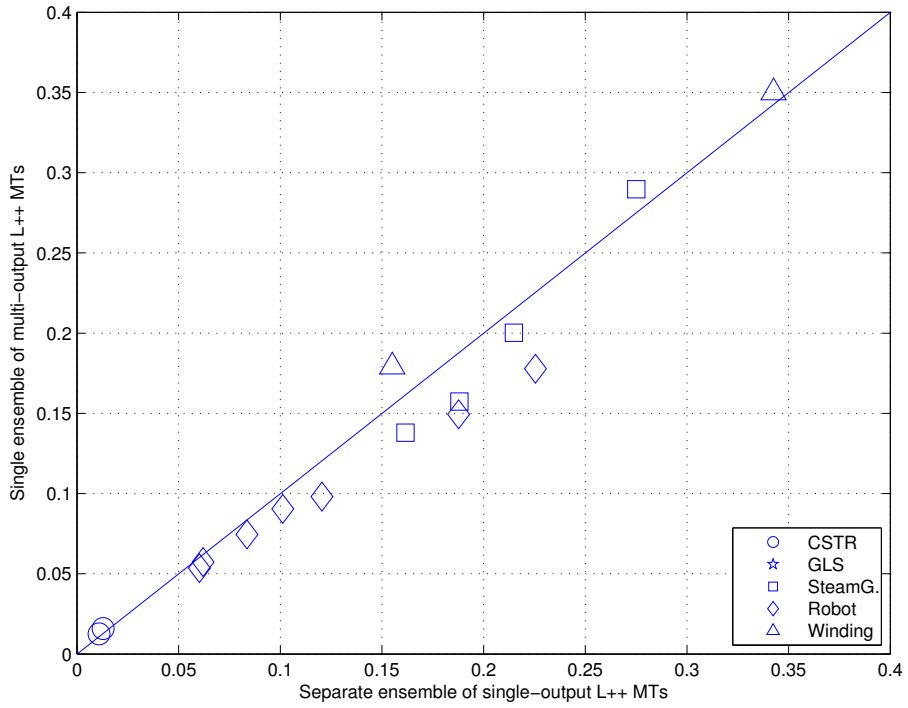


Figure 6.31: The predictive performance of separate bagging of single-output L++ model trees, one for each output, to a bagging model which utilizes multi-output L++$_{MO}$ model trees.

6.8 Summary

In summary, the evaluation of the two types of model tree algorithms, and ensembles thereof, showed differing results, which were in favor of the soft model tree formalism.

The analysis with the general-purpose model tree algorithm M5' did not yield conclusive results regarding the minimal modifications for successful application for modeling dynamic systems. The conclusions that the analysis provided regarding this were:

- The output error results of single M5' trees showed larger errors as compared to Lolimot, i.e., a soft tree approach.

- The post-smoothing procedure decreased the output error only slightly. Only in a few cases it helped decrease the error substantially. In our opinion it does not solve the problem of large output errors.
- The post-pruning procedure overpruned the tree for many of the case studies, thus deteriorating the output error performance.
- Randomization of the split attribute as included in the Forests method helped. It showed decreased errors as compared to bagging.
- Ensemble selection of ensembles of M5' model trees helped. The M5' model trees created in an ensemble showed different performance. It is a fact that some of the trees, i.e., base models in the ensemble, are not appropriate for modeling the dynamical system and show large output errors, or in the worst case the simulation of such trees diverges. In our opinion the ensemble selection procedure, which utilizes the output error on the training set removes the trees which are not appropriate. This in turn increases the performance of the resulting reduced ensemble, over the full ensemble.
- The results for the Robot dataset, which consists of the largest number of input variables were not comparable to that of Lolimot.

Regarding the modifications required for a general-purpose crisp model tree algorithm, we present several guidelines and discuss further work. The replacement of the crisp local model estimation with soft would result in a more robust estimation procedure. This naturally assumes that the soft formalism of Eq. (4.4) would be chosen too. A lookahead procedure would be more useful for a general-purpose crisp model tree algorithm, however, the split cut-point needs tuning, which can be computationally demanding. The post-pruning phase of a general-purpose tree learning algorithm might be replaced with a pre-pruning phase.

The analysis also revealed several different properties of Lolimot and ensembles of Lolimot. In particular, the benefits over a crisp approach, for modeling dynamic systems, are several:

- The fuzzy estimation of local model parameters is more robust than the crisp - it uses all available data, and it can produce estimates even for partitions which have none, or only a few training data points.
- The natural type of evaluation of a model the tree of the previous point is by using interpolation of the local models (treating it as a soft model tree).
- The model predictions are smooth, and as such are more appropriate to model the static nonlinearity (external dynamics approach).
- The optimization of the split cut-point is not as influential to the model performance, as in the crisp case.

The empirical analysis of the improved version of the Lolimot algorithm, named L++, showed decreased running times, while the output error was similar to that of Lolimot. The tuning of the overlap parameter proved to be useful for obtaining models with better predictive performance. The bagging of soft model trees, which was also evaluated, showed improved prediction results. The difference was visible both when comparing Lolimot model trees to bagging of Lolimot in Subsection 6.6.1, and when comparing L++ model trees to bagging of L++ in Subsection 6.6.2.

The analysis of the multi-output problems in Subsection 6.7.2 showed that the learning of multi-output trees is faster than several single-output trees. Also, when the model is evaluated using simulation, the single-output model trees tend to overfit to the training data and produce a diverging simulation result. The multi-output learning of trees has a clear advantage here, since the candidate splits are selected based upon the performance of the parallel simulation, i.e., a simulation for all output variables. Also, the diverging single-output models appeared on a noisy version of the synthetic CSTR dataset, and on the measured GLS dataset, whose measurements are not ideal (at least for the h_1 variable). Both noisy and non-ideal measurements are properties of realistic modeling problems that the user might come across.

Chapter 7

Conclusions

7.1 Summary and Discussion

This thesis considered the problem of discrete-time modeling of dynamic systems, by using black-box modeling techniques. In particular, it considered the crisp model tree approaches introduced in the machine learning domain and the soft/fuzzy linear model tree approaches, or Takagi-Sugeno fuzzy models, introduced in the system identification domain. The thesis also considered ensembles of both types of linear model trees.

The discrete-time modeling was reformulated, according to the external dynamics approach, into a static nonlinear function approximation problem. This thesis studied and evaluated whether the local linear model trees and ensembles are suitable for solving the static approximation task, and eventually, provide an appropriate discrete-time model of the dynamic system. These models are learned from measured input-output data, measured at discrete time intervals, and provide an input-output mapping of the studied system.

The thesis considered models that describe closely the studied real-world phenomena. Typically, two objectives are of importance when pursuing the modeling task: good predictive performance and interpretability of the models. In the thesis, we studied and analyzed mainly the predictive performance objective. However, the models built in the form of model trees also allow for some kind of interpretation (cf. Subsection 4.4.1), which can be obtained by looking at the local model coefficients. Both the crisp and the fuzzy local estimation provide for interpretable coefficients, representing local linearizations around the operating point of the nonlinear dynamic system.

The thesis introduced and evaluated several modifications to the general purpose model-tree learning algorithm M5'. It tried to determine a minimal set of properties that the crisp M5' algorithm needs to have, in order to be appropriate for modeling dynamic systems, either in an ensemble or a single tree setting. The modifications were motivated by the design of the Lolimot method, and in particular its fuzzy local model estimation and the type of models it produces. Subsection 6.5.1 provided an empirical comparison of the differences between the crisp estimation of M5' and the fuzzy local estimation of Lolimot. In particular, it evaluated a modification of M5' which replaced the crisp trees and local model estimation with fuzzy local model estimation. It concluded that the fuzzy local estimation and fuzzy model tree formalism are able to improve the predictive performance of M5'.

Two conclusions were made regarding the modifications of M5': On the one hand, the use of the fuzzy local model estimation for unpruned trees would require that local models are estimated for trees with a large number of terminal nodes. This is why the crisp LM estimation in M5' was retained. On the other hand, the fuzzy model tree formalism was successfully utilized: The crisp M5' model tree as built by the original method was

converted to a fuzzy model tree by a procedure of fuzzification. This procedure did not re-estimate the local models of the tree, and only converted the crisp splits into fuzzy ones, transforming the crisp model tree into a fuzzy model tree.

However, as the empirical analysis showed, the modifications introduced to M5' did not yield conclusive results regarding the minimal set of properties for its successful application for dynamic system identification. The predictive performance of both single trees built by the improved M5' and ensembles of improved M5' tree was still lower than that of Lolimot and ensembles of Lolimot. Also, the analysis showed that the M5' model tree algorithm may identify incorrect models. For some datasets, the crisp local model estimation was performed on a small number of data points, which led to identification of incorrect local models. These in turn contained overly large or small coefficients, which resulted in large overall model errors. The culprit for this is the heuristic split selection, which we discuss below.

It is our opinion that the heuristic split selection procedure of M5' should be altered, as for example, it had trouble obtaining good results for all of the outputs of the Robot case study, which contains a large number of input variables. In such cases, we suggest that instead of replacing the heuristic split selection with a look-ahead approach which uses crisp LM estimation, the user resorts to a look-ahead approach which uses the soft LM estimation. By doing so, the user would make use of a more powerful formalism, with potentially lower computational complexity, as the choice of the cut-points in the splits was shown to have almost no influence on the error of the final model.

The thesis also provided an in-depth evaluation of the properties of Lolimot and considered several ways of modifying it. The evaluation showed that the soft model tree approach Lolimot produced models with lower output error as compared to the crisp general-purpose M5'. Additionally, as the number of iterations was limited to 30, the models produced were also smaller as compared to M5'. The analysis concluded that two improvements to Lolimot are beneficial: a) altering its model evaluation in the look-ahead split selection phase, and b) altering the structure of the model trees by tuning the overlap of the fuzzy membership functions (MSFs).

The advantages of the first modification are decreased running times of the method, while the models built preserved their predictive performance. The advantage of the second are the slightly lowered output errors, which were, however, not statistically significant according to the the Wilcoxon test at the 1% level. The conclusion made here is that although the overlap in Lolimot is calculated by using a fixed k_σ value (Nelles, 1999), the empirical analysis showed that the amount of overlap k_σ, of the fuzzy MSFs, needs tuning for each dataset.

The thesis provided two empirical analyses of the improved model tree algorithms and ensembles thereof: a) on static regression tasks, and b) on dynamic system case studies. In particular, the analysis on the static regression tasks showed that the forests of M5' model trees increased the predictive performance over single M5' model trees. The randomness introduced in the split selection of the M5' method in the forest setting improved the results, both for single-target and multi-target regression.

The results also showed that the forests of model trees improved upon forests of regression trees. This means that one might expect potentially good predictive performance by ensembles of model trees. Also, the analysis on the static regression tasks compared single trees and concluded that Lolimot and M5' show similar performance, and that they improve over the performance of a single regression tree.

The analysis on the dynamic system case studies compared ensembles of model trees built by the improved M5', and the improved Lolimot. Regarding the former, the forests of M5' model trees with ensemble selection provided satisfactory predictive performance.

The ensemble selection procedure removed the trees which were not successful from the ensemble and increased its predictive performance. Also, similar to the static case, the randomization of the base learning algorithm, as implemented in the forests of M5' model trees, showed improved performance over bagging and single M5' trees.

Regarding the latter, the bagging of Lolimot trees improved the performance over a single Lolimot tree. The improvement, however, was not statistically significant at the 1% level, due to the increased error on a few case studies. Finally, the comparison to selected methods, typically used for modeling dynamic systems, showed that bagging of improved Lolimot trees produced slightly better predictive performance results over ANFIS and Neural Networks. The improvement over ANFIS was rather unexpected, and was found to be due to the suboptimal parameters for ANFIS, chosen by using the same validation set as the other methods.

In summary, the thesis evaluated ensembles of both crisp and soft model trees and concluded that soft model trees and ensembles thereof can solve the discrete-time modeling problem of nonlinear dynamic systems, while the crisp approaches can provide satisfactory results in many cases, by using the forest ensembles with ensemble selection. The ensembles of soft Lolimot model trees are a more accurate and slower alternative, as compared to ensembles of crisp M5' model trees, which are less accurate and faster. The ensembles of crisp M5' model trees improved the performance over a single model tree, and also the ensembles of soft Lolimot model trees improved the performance over a single Lolimot tree.

The thesis also studied the modeling of multi-input multi-output (MIMO) dynamic systems by using multi-output (multi-target) model trees and ensembles of multi-output trees. It started with the typical approach to solving the multi-output problem, which builds single-output models for each of the output variables, i.e., breaking up the MIMO problem into several MISO problems. It evaluated an alternative multi-output tree approach which builds local models for all output variables in each of the terminal nodes of a single multi-target tree. The inter-dependencies between the outputs and the "common directions of nonlinearity" would potentially enable building multi-output trees with smaller total size (number of splits) and almost equal performance, as compared to the former approach.

The empirical analysis showed that the multi-output model trees are a more promising approach. Two advantages of multi-output model trees were noticed: a) the learning of a multi-output tree is faster than a set of single-output ones, and b) the single-output model trees might overfit to the training data and produce incorrect models. The latter advantage of the multi-output variant is due to the candidate split selection procedure: it uses the intermediate models for each output in a parallel simulation procedure. The latter is more appropriate for detection of incorrect multi-output models. Also, the parallel simulation procedure provides the final model assessment, and using it during learning is beneficial for the multi-output model.

7.2 Scientific Contribution

The research presented considered two linear model tree learning algorithms, introduced in different communities. The thesis compared their similarities and differences, evaluated them for modeling dynamic systems, and proposed modifications for each of them. The work presented here also studied and evaluated ensembles of model trees. It took into account the modeling of both single-output and multi-output dynamic systems, by multi-output model trees, or multi-output Takagi-Sugeno models. Additionally, it evaluated the model tree ensembles on machine learning benchmark regression tasks, and concluded that the ensembles provide an increase in performance over single model trees. The main

contributions of the thesis can be summarized as follows.

1. Design and implementation of novel model-tree based approaches for modeling dynamic systems, based on and improving upon the M5' and Lolimot algorithms.

a. Improved M5' algorithm for regression, which can now induce fuzzified and multi-target model trees. We analyzed the benefit of fuzzy local model estimation, and the fuzzy model tree formalism, applied to an M5' model tree. Since the fuzzy estimation is more computationally expensive than the crisp, we introduced a fuzzyfication step after the crisp M5' model tree is learned. This step preserves the local models already estimated, and only converts the crisp tree to a fuzzy tree, by replacing the crisp splits with fuzzy ones.

We also introduced multi-target M5' model trees. Modifications to the heuristic split selection as well as to the pruning procedure were made. The terminal nodes contain a linear model for each of the targets, with each estimated independently of the other.

b. Improved Lolimot algorithm for modeling dynamic systems, which now produces trees with similar predictive performance faster. The work in this thesis improved the Lolimot algorithm by a) introducing a faster model evaluation procedure and b) by optimizing the fuzzy overlap. The improved Lolimot, with these two modifications, produces model trees with similar performance and has reduced running time.

This thesis also considered an optimization of the split cut-points in Lolimot, which did not provide an improvement in the predictive performance. In more detail, the results of the analysis showed that optimizing the cut-points while keeping the fuzzy overlaps fixed does not improve predictive performance. This means that the cut-points and overlaps should probably be optimized simultaneously.

c. Algorithms that can induce ensembles of single and multi-target model trees by using the improved M5' and Lolimot algorithms. This work introduced the MTE algorithm which induces ensembles based on the bagging and forests of the improved model tree algorithm M5'. The MTE algorithm can be used for single and multi-target regression tasks, as the fuzzified M5' model trees support this. MTE also contains an ensemble selection approach which reduces the size of the ensemble, by removing some of the model trees. It uses the output error for evaluation of the intermediate ensembles. The ensemble selection approach has the potential to build ensembles with similar or improved predictive performance, while using a smaller number of trees.

This work also introduced ensembles of Lolimot and the improved Lolimot algorithms. The ensembles use the bagging approach. Both MTE and ensembles of Lolimot can be used for single- and multi-target problems.

2. Empirical evaluation of the developed approaches on benchmark problems and case studies.

a. Evaluation of the improved M5' and Lolimot algorithms (and ensembles based on these) on benchmark problems of single and multi-target static regression. The thesis reported the results of an empirical evaluation of the improved M5' and Lolimot, and ensembles thereof, on regression problems typically considered in the machine learning domain. The predictive performance of the M5' linear model trees was increased by using forests of such trees, the same applies for bagging ensembles of Lolimot model trees. The introduction of randomness in the split selection procedure in M5' proved to be beneficial in the ensemble setting, both for single-target and multi-target problems. The empirical evaluation of M5' showed that the crisp local model estimation may fail to detect matrix singularity issues and as a result, identify incorrect local models. These incorrect local models, might contain overly large or small coefficients and produce models with large errors. The soft local model estimation might be considered here instead, as it is more robust against this issue.

b. Evaluation of all the above approaches (and a few other selected methods) on several case-studies of modeling dynamic systems. The thesis reported the results of an empirical evaluation of all approaches on datasets derived from several dynamic system case studies. It evaluated the improved Lolimot, bagging of Lolimot model trees and bagging and forests of improved M5' model trees. The bagging of Lolimot model trees improved the predictive performance over a single Lolimot model tree, however the running times were obviously increased. The best prediction result using M5' model trees was obtained by using the MTE algorithm, i.e., forests of M5' model trees, with ensemble selection. The latter helped increase the predictive performance by removing the incorrect M5' trees from the ensemble. However, the predictive performance of forests of M5' was significantly worse than the one of bagging of improved Lolimot model trees. Also, the comparison of bagging of improved Lolimot trees to a few other selected methods used in system identification showed slightly better predictive performance.

7.3 Further Work

We would like to consider three major directions for further work. First, we would like to apply the methods we have developed to additional case studies, especially case studies of natural (as opposed to man-made) dynamic systems. Second, we would like to further improve the methods developed here. Finally, we would like to consider the development of new or adaptation of other machine learning techniques (not considered so far) for solving the problem of modeling dynamic systems.

Ecosystems. The first direction for further work considers the (single- and multi-output) modeling of ecological dynamic systems, such as lake ecosystems. Preliminary experiences in applying the methods from this thesis to such domains show that the measurements in these domains are far from perfect: not all variables which influence the outputs are measured, and the sampling time used to obtain the measurements is too coarse, i.e., not short enough. We believe that improving the quality of the data for such systems may lead to successful modeling of these systems by using discrete-time approaches, such as the ones presented in this thesis.

Further improvements. To improve upon the approaches considered here, we would like to evaluate the potential of soft global estimation of the local model parameters in Lolimot. As shown in the empirical evaluation, when combined with linear model trees, such estimation leads to smaller trees with increased predictive performance. However, the optimal number of local models is hard to determine, because soon after reaching the optimal number of local models, the predictive performance of such model trees quickly deteriorates, due to overfitting. Also, the coefficients identified by global estimation do not allow the local models to be interpreted as local linearizations of the nonlinear dynamic system, in case this is a requirement of the modeling. Resolving these two issues is likely to be a significant challenge.

Further development. In terms of further development of machine learning methods for modeling dynamic system in discrete time, we would like to consider the use of background knowledge, the adaptation of boosting and the use of methods for online learning, or learning from data streams.

Regarding the use of background knowledge that the user might have about the system, the model trees allow for some knowledge of the system to be included in the model. Two options can be used: a) pre-determining the first several split nodes, which would influence the positioning of the MSFs, and b) pre-determining the subsets of regressors that are to be used in specific parts of the operating regions. While option a) has been explored in multi-target regression trees, option b) and the combination of the two options remain to

be explored.

The boosting of model trees might yield ensembles with a smaller number of trees, as compared to bagging. These, however cannot be built in parallel, as is the case with bagging. Also, the Lolimot algorithm may have to be modified to consider more splits in each dimension (as proposed by one of its modifications in the thesis), or add randomization in the split selection procedure. Namely, the current split selection procedure might lead to positions and overlaps of membership functions that are very similar in the first few Lolimot iterations, and to local models whose coefficients are estimated by using the same data points and weights both for the first and the subsequent steps of boosting.

Approaches that learn incrementally, in an online manner, allow for modeling of time-varying dynamic systems. They can detect changes in the sequential/streaming data and adapt the model accordingly. This can be used for discovering potential faults or changes in the dynamic system being monitored. Some online approaches based on model trees have already been introduced (Nelles, 1996; Potts & Sammut, 2007).

The same holds for techniques (e.g., (Ikonomovska, Gama, & Džeroski, 2011)) for learning data streams that can be applied to large quantities (data streams) of sequential data that arrive constantly and endlessly at very high rates and cannot be stored for longer periods of time. Such data for learning from data streams can be difficult/impossible to handle using the standard offline techniques, due to time and memory constraints. Evaluation using the output error is typically not used in the online setting, but approaches (such as the ensembles of improved Lolimot trees), which use the output error during learning are potentially more successful at the identification of dynamic systems: using output error when modeling dynamic systems with model trees or ensembles on data streams (Ikonomovska, 2012) is thus a challenge worth addressing.

Appendix A

Complete Results

This appendix presents the complete results for the empirical analysis a) on the benchmark machine learning datasets (Chapter 4), and b) for modeling dynamic systems (Chapter 5). In particular:

- Tables A.1 to A.10 show the complete results in the static case,
- Tables A.11 to A.26 show the complete results of the evaluation for modeling dynamic systems.

Table A.1: A statistical comparison of the predictive performance of different tree learning algorithms for the task of single-target regression. The results in all tables compare the leftmost method, in this case M5' MT, to all of the other methods, by using paired comparisons. Additionally, the comparison signs $<, =, >$ indicate the result of the paired comparison according to the t-test.

	M5' MT:	t-test	Lolimot	t-test	RT
abalone	0.6667	=	0.6569	=	0.7064
analcat	0.1422	=	0.1402	=	0.1478
auto93	0.5807	<	1.0928	=	0.6467
autoMpg	0.3693	=	0.3736	<	0.4417
auto-price	0.3971	=	0.4375	=	0.3990
bank8FM	0.2010	=	0.2009	<	0.2480
baseball	0.5818	=	0.5710	=	0.6255
baskball	0.8086	=	0.7670	=	0.9089
bodyfat	0.1545	=	0.1611	=	0.1864
breastTumor	0.9613	=	0.9706	=	1.0009
cal-housing	0.4668	<	0.5641	<	0.5147
cholesterol	1.0118	=	1.0343	=	1.0447
cleveland	0.7024	=	0.6910	<	0.8493
cloud	0.3857	=	0.4007	=	0.5518
concrete	0.3454	=	0.3429	<	0.4262
cpu	0.2356	=	0.1784	=	0.4028
cpu-act	0.1446	>	0.1228	<	0.1744
dailyElectrEner	0.4385	=	0.4184	<	0.4897
delta-ailerons	0.5446	=	0.5420	<	0.5831
delta-elevators	0.6969	>	0.6822	=	0.7025
echoMonths	0.7136	=	0.7388	=	0.7364
electr-len-2	0.0407	<	0.0519	<	0.0556
fishcatch	0.1737	=	0.1433	=	0.2229
forestFiresPOR	1.0196	=	1.0011	=	1.1849
fruitfly	1.0230	=	1.0449	=	1.0000
housing	0.4031	=	0.3751	<	0.5034
hungarian	0.7450	=	0.7428	<	0.8410
kin8nm	0.5857	>	0.5654	<	0.6909
laser	0.2080	>	0.1626	=	0.2328
lowbwt	0.6433	=	0.6366	=	0.6318
machine-cpu	0.3251	=	0.2902	=	0.3857
meta	0.8127	=	1.0062	=	0.9663
mortgage	0.0417	=	0.0205	<	0.0762
pbc	0.7957	=	0.8260	<	0.9096
pharynx	0.3577	<	1.3477	=	0.3730
pol	0.1485	<	0.4404	<	0.1960
puma8NH	0.5712	>	0.5627	<	0.5943
pwLinear	0.3102	=	0.3120	<	0.4481
quake	0.9958	=	1.0014	=	1.0016
sensory	0.8459	=	0.8437	<	0.9323
servo	0.3510	=	0.2410	=	0.3771
stock	0.1378	=	0.1262	=	0.1638
strike	1.0743	=	0.9209	=	1.0239
treasury	0.0732	=	0.0576	<	0.1044
triazines	7.2210	=	1.0973	=	0.8973
veteran	0.9652	=	1.1412	=	0.9383
wankara	0.0825	>	0.0785	<	0.1283
wisconsin	0.9905	=	0.9809	=	1.0063
wizmir	0.0834	>	0.0778	<	0.1207
t-test			5:7		21:0
w-test			0.384		0.000

Table A.2: A statistical comparison of the tree sizes in terms of the number of local models, and the running times. Three different tree learning algorithms evaluated for the task of single-target regression. In this and the following tables which report model sizes and running times, the model sizes are expressed as an average number of terminal nodes of the 10 folds, while the running times are expressed as a sum of the total time required for learning.

	Model size			Learning time (sec.)		
	M5' MT:	Lolimot	RT	M5' MT:	Lolimot	RT
abalone	12.0	7.7	36.2	574.9	12187.8	148.4
analcat	80.2	20.2	12.0	113.6	11072.8	57.2
auto93	39.0	8.5	7.0	32.4	365.5	12.5
autoMpg	4.4	3.5	16.0	51.9	426.6	23.5
auto-price	62.4	3.9	62.3	22.3	195.0	10.7
bank8FM	45.0	29.9	255.0	746.2	67071.5	450.2
baseball	6.4	3.6	10.0	35.2	223.7	14.9
baskball	1.5	2.1	2.1	12.5	26.0	5.4
bodyfat	33.0	4.4	33.0	19.0	133.2	8.2
breastTumor	1.0	1.0	2.0	44.7	310.8	22.0
cal-housing	7645.0	6.0	499.0	1901.6	9283.2	1039.2
cholesterol	2.0	1.2	4.0	61.0	247.0	18.1
cleveland	1.0	1.0	6.0	27.8	383.4	13.1
cloud	1.0	1.3	43.0	12.1	30.4	7.6
concrete	404.0	28.6	404.0	69.4	1159.9	35.0
cpu	4.0	5.8	34.0	24.3	357.6	10.4
cpu-act	8.0	27.4	270.0	1056.2	103635.8	969.4
dailyElectrEner	1.6	9.4	8.0	25.8	107.1	11.1
delta-ailerons	22.0	28.9	74.0	478.9	35770.4	238.3
delta-elevators	8.0	24.2	59.0	577.0	66766.8	306.4
echoMonths	1.0	1.2	2.0	13.7	41.9	6.5
electr-len-2	50.6	29.7	113.0	30.7	816.5	31.1
fishcatch	3.0	4.6	41.0	18.6	63.8	7.1
forestFiresPOR	119.0	1.1	1.0	65.2	564.9	24.2
fruitfly	1.0	1.4	1.0	13.5	29.9	5.8
housing	193.0	16.2	193.0	48.2	245.8	25.2
hungarian	1.0	1.5	4.0	23.8	317.8	10.5
kin8nm	1632.0	30.0	264.0	863.9	66828.9	432.8
laser	49.0	27.0	219.0	47.1	642.9	20.5
lowbwt	1.0	1.1	2.0	21.7	98.4	9.5
machine-cpu	62.0	3.8	62.0	14.4	45.6	7.7
meta	10.0	8.8	47.0	108.0	2337.0	29.0
mortgage	99.7	23.8	99.0	79.1	1868.2	34.0
pbc	1.0	1.1	6.0	59.7	297.0	20.3
pharynx	5.0	15.4	14.0	250.9	15239.3	73.9
pol	1305.0	6.0	1305.4	1832.8	15998.4	1287.6
puma8NH	35.0	22.2	83.0	908.6	56952.5	503.3
pwLinear	2.0	2.5	14.0	18.2	124.2	11.2
quake	1.0	9.1	1.0	123.5	1329.7	58.7
sensory	247.0	2.0	8.0	98.0	734.4	33.8
servo	37.0	3.3	49.0	16.4	86.3	7.9
stock	253.4	29.7	253.3	71.4	1175.5	30.6
strike	1.0	1.1	10.0	67.7	683.6	32.3
treasury	34.0	15.7	59.0	73.5	1623.4	30.8
triazines	77.0	3.4	5.0	120.2	825.5	22.6
veteran	1.0	1.9	2.0	15.5	42.2	8.8
wankara	4.0	19.7	415.3	120.6	2249.4	52.9
wisconsin	1.0	1.0	3.0	65.9	300.5	13.4
wizmir	2.0	13.4	345.1	92.6	1722.8	44.5
#wins[1]		23:26	42:7		49:0	1:48
w-test		0.003	0.019		0.000	0.000

[1]The number of wins, denoted as "#wins" is reported in this and in the following tables with results for the size of the models and the running time. The values only summarize the number of datasets on which variant A had a smaller value than variant B. No statistical test is considered. The sum of the number of wins for the method tested and its alternative would always add up to the total number of datasets.

Table A.3: A statistical comparison of the predictive performance of different tree learning algorithms, for the task of multi-target regression.

	M5' MT:	t-test	Lolimot	t-test	RT
class-spec	6.7591		2.5555		0.8727
class-ind	5.7335		6.1634		0.8540
class-fols	0.9345		3.5040		0.9435
Collembola		=		=	
DFlow	0.7828		0.8094		0.9106
DGap	0.7127		0.7208		0.7667
EDM		=		=	
p	0.0350		0.0320		0.1113
aspect	0.9866		0.9880		1.0000
Forestry IRS		=		<	
p	0.0334		0.0296		0.1113
aspect	0.9962		0.9957		1.0000
Forestry SPOT		=		<	
MF0/00 plR	0.5788		0.8406		0.6190
MS0/00 plR	0.5447		0.6420		0.6481
Sigmea-real		=		=	
Dispersal Rate Pollen	297.7392		0.0336		0.0429
Dispersal Seeds	228.3116		0.0257		0.0224
Sigmea-simulated		=		=	
c-class	0.9523		0.9554		0.9649
m-class	0.9237		0.9284		0.9686
x-class	0.9584		0.9443		1.0087
Solar-flare1		=		<	
c-class	0.8761		0.8922		0.8935
m-class	0.9859		0.9785		0.9883
x-class	0.9711		0.9957		0.9747
Solar-flare2		=		=	
Cladophora	0.9652		0.9359		1.0099
Gongrosira	0.9943		0.9773		1.0177
Oedogonium	0.9047		0.9039		0.9499
Stigeoclonium	0.8729		0.8927		0.9116
Melosira	0.9507		0.9613		0.9819
Nitzschia	0.8235		0.8231		0.8421
Audouinella	0.8664		0.8586		0.8967
Erpobdella	0.9113		0.9764		0.9356
Gammarus	0.8110		0.8250		0.8381
Baetis	0.9386		0.9416		0.9741
Hydropsyche	0.9468		0.9749		0.9955
Rhyacophila	0.8561		0.8841		0.8707
Simulium	0.9918		0.9939		1.0303
Tubifex	0.8765		0.8931		0.9084
Water quality		=		<	
t-test			0:0		4:0
w-test			0.150		0.006

Table A.4: A statistical comparison of the tree sizes in terms of the number of local models, and the running times. Three different tree learning algorithms evaluated, for the task of multi-target regression.

	Model size			Learning time (sec.)		
	M5' MT:	Lolimot	RT	M5' MT:	Lolimot	RT
Collembola	1.1	1.0	2.0	626.1	6232.8	133.0
EDM	7.0	1.8	13.0	16.1	159.9	6.2
Forestry IRS	1.0	1.4	1.0	710.2	10932.7	371.0
Forestry SPOT	1.0	2.0	1.0	1840.2	20076.3	434.6
Sigmea-real	49.2	2.7	49.2	19.7	272.1	15.9
Sigmea-simulated	30.4	28.8	36.0	297.8	62192.4	213.0
Solar-flare1	12.1	1.0	1.0	21.9	568.6	13.3
Solar-flare2	13.2	1.1	13.2	95.2	2166.5	25.7
Water quality	53.5	2.8	53.5	1005.7	2614.4	279.6
#wins		3:6	8:1		9:0	0:9
w-test		0.039	0.875		0.004	0.004

Table A.5: A statistical comparison of the predictive performance of ensembles, for the task of single-target regression.

	Forests MT :	t-test	Bagg.MT	t-test	MT	t-test	Forests RT	t-test	Bagg.RT	t-test	Bagg.Lol
abalone	0.6535	=	0.6487	=	0.6667	=	0.6680	=	0.6695	=	0.6527
analcat	0.1474	=	0.1428	=	0.1422	=	0.1463	=	0.1473	=	0.1406
auto93	0.4279	=	0.4441	=	0.5807	=	0.4949	<	0.5030	<	0.6700
autoMpg	0.3286	=	0.3391	=	0.3693	=	0.3402	=	0.3531	=	0.3516
auto-price	0.4101	=	0.3975	=	0.3971	=	0.3482	=	0.3677	=	0.4258
bank8FM	0.1895	<	0.1936	<	0.2010	<	0.2056	<	0.2075	<	0.2001
baseball	0.5496	=	0.5552	=	0.5818	=	0.5675	=	0.5729	=	0.5494
baskball	0.8035	=	0.8004	=	0.8086	=	0.8932	=	0.8627	=	0.7687
bodyfat	0.1591	=	0.1502	=	0.1545	=	0.1590	=	0.1703	=	0.1558
breastTumor	0.9558	=	0.9541	=	0.9613	=	0.9637	=	0.9637	=	0.9704
cal-housing	0.4236	<	0.4305	<	0.4668	=	0.4220	=	0.4225	<	0.5409
cholesterol	1.0026	=	1.0095	=	1.0118	=	0.9902	=	0.9923	=	1.0312
cleveland	0.6925	=	0.6996	=	0.7024	=	0.7014	=	0.7406	=	0.6909
cloud	0.3953	=	0.3788	=	0.3857	=	0.4817	=	0.4552	=	0.3896
concrete	0.3006	<	0.3168	<	0.3454	=	0.2978	=	0.3057	<	0.3311
cpu	0.1613	=	0.1543	=	0.2356	=	0.3480	<	0.3331	=	0.1403
cpu-act	0.1265	=	0.1357	<	0.1446	<	0.1329	<	0.1400	>	0.1198
dailyElectrEner	0.4065	=	0.4134	=	0.4385	=	0.4156	=	0.4273	=	0.4140
delta-ailerons	0.5266	=	0.5290	<	0.5446	<	0.5388	<	0.5439	<	0.5445
delta-elevators	0.6734	=	0.6806	=	0.6969	=	0.6552	=	0.6648	=	0.6814
echoMonths	0.7189	=	0.7184	=	0.7136	=	0.7141	=	0.7147	<	0.7383
electr-len-2	0.0398	=	0.0410	=	0.0407	<	0.0490	<	0.0481	<	0.0506
fishcatch	0.1362	=	0.1396	<	0.1737	<	0.1940	<	0.1928	=	0.1418
forestFiresPOR	1.0080	=	5.5383	=	1.0196	=	1.0050	=	1.0224	=	1.0008
fruitfly	1.0216	=	1.0340	=	1.0230	=	1.0000	=	1.0021	=	1.0441
housing	0.3648	=	0.4285	=	0.4031	=	0.3723	=	0.4045	=	0.3581
hungarian	0.7195	=	0.7183	=	0.7450	=	0.7691	=	0.7694	=	0.7364
kin8nm	0.6995	=	0.5261	=	0.5857	=	0.5067	=	0.5131	=	0.5659
laser	0.1328	=	0.1507	<	0.2080	<	0.1776	=	0.1882	<	0.1572
lowbwt	0.6178	=	0.6272	=	0.6433	=	0.6133	=	0.6173	=	0.6395
machine-cpu	0.2863	=	0.2892	=	0.3251	=	0.3785	=	0.3603	=	0.2886
meta	0.6285	=	0.8312	=	0.8127	=	0.7160	=	0.7334	=	0.7713
mortgage	0.0221	=	0.0259	=	0.0417	<	0.0509	<	0.0568	=	0.0204
pbc	0.8042	>	0.7938	=	0.7957	=	0.8190	=	0.8427	=	0.8234
pharynx	0.2974	=	0.2987	=	0.3577	=	0.2915	=	0.2958	=	0.4941
pol	0.1035	<	0.1228	<	0.1485	<	0.1222	<	0.1426	<	0.2454
puma8NH	0.5638	=	0.5650	<	0.5712	<	0.5685	<	0.5701	<	0.5717
pwLinear	0.3441	=	0.3246	=	0.3102	=	0.3645	=	0.3707	=	0.3080
quake	1.0232	=	1.0103	=	0.9958	=	0.9974	=	0.9966	=	1.0003
sensory	0.8237	=	0.8252	<	0.8459	=	0.8283	=	0.8332	=	0.8438
servo	0.2721	=	0.2915	<	0.3510	=	0.3478	<	0.3190	=	0.2466
stock	0.1049	=	0.1180	<	0.1378	<	0.1115	=	0.1229	<	0.1233
strike	0.9067	=	0.9064	=	1.0743	=	0.9159	=	0.9201	=	0.9218
treasury	0.0534	<	0.0622	<	0.0732	<	0.0668	<	0.0750	=	0.0558
triazines	0.8049	=	0.8173	=	7.2210	=	0.7825	=	0.7745	=	0.9345
veteran	0.9493	=	0.9475	=	0.9652	=	0.9562	=	0.9445	=	0.9880
wankara	0.0766	<	0.0797	<	0.0825	<	0.0864	<	0.0903	=	0.0775
wisconsin	0.9488	=	0.9409	<	0.9905	=	0.9821	=	0.9758	<	0.9904
wizmir	0.0765	<	0.0793	<	0.0834	<	0.0842	<	0.0884	=	0.0781
t-test			7:1		16:0		13:0		14:0		12:1
w-test			0.028		0.000		0.001		0.000		0.003

Table A.6: A statistical comparison of the model sizes and running times of ensembles, for the task of single-target regression.

	Model size						Learning time (sec.)					
	Forests MT :	Bagg.MT	MT	Forests RT	Bagg.RT	Bagg.Lol.	Forests MT:	Bagg.MT.	MT	Forests RT	Bagg.RT	Bagg.Lol.
abalone	23.5	11.5	12.0	1569.0	85.5	16.0	327840.8	82852.4	574.9	21000.0	5468.9	68616.6
analcat	55.5	85.5	80.0	62.0	21.5	12.4	194682.0	51574.2	113.6	3232.5	1509.6	13254.1
auto93	1.0	1.0	39.0	37.5	32.5	8.5	13216.0	3707.1	32.4	450.9	206.0	8629.6
autoMpg	155.0	3.5	4.0	155.0	153.5	3.5	31505.0	81626.4	51.9	1630.1	604.1	8394.0
auto-price	8.5	7.5	62.0	54.0	53.5	3.9	11717.0	2974.3	22.3	497.7	175.5	2282.7
bank8FM	2377.0	2207.0	45.0	2277.5	2207.0	28.9	2456439.2	176910.8	746.2	31342.6	21441.4	48926.9
baseball	2.5	3.5	6.0	111.5	16.5	3.6	51681.8	5376.5	35.2	951.0	360.1	4970.0
baskball	1.0	1.0	1.0	39.5	3.5	2.1	8457.8	2230.5	12.5	395.9	105.6	294.7
bodyfat	63.0	33.5	33.0	46.0	33.5	4.4	32968.0	7162.2	19.0	613.2	221.2	3697.3
breastTumor	1.5	1.5	1.0	3.0	2.0	1.0	199158.5	73252.1	44.7	1491.4	509.0	7065.6
cal-housing	7278.5	6940.0	7645.0	6989.0	6940.0	29.9	1749791.5	575609.2	1901.6	88797.7	122594.1	79924.4
cholesterol	1.0	2.5	2.0	6.5	6.0	1.2	24521.2	6718.6	61.0	809.3	413.7	4037.9
cleveland	1.0	1.0	1.0	83.5	69.5	1.0	16164.2	4156.4	27.8	676.1	329.5	4074.8
cloud	2.5	1.0	1.0	41.0	39.0	1.3	8873.0	2095.7	12.1	294.4	88.0	509.6
concrete	32.0	47.5	404.0	369.0	352.0	27.8	378604.4	19148.7	69.4	4022.9	823.9	22866.9
cpu	17.0	6.5	4.0	31.5	29.5	5.8	11454.6	10196.4	24.3	636.5	220.1	8983.5
cpu-act	1927.5	1868.5	8.0	1927.5	1868.5	28.7	680543.6	227160.8	1056.2	40114.6	19331.3	380124.6
dailyElectrEner	74.5	77.0	1.0	141.0	137.5	9.4	27731.8	6766.0	25.8	1191.9	388.9	2913.4
delta-ailerons	2522.0	2305.5	22.0	2522.0	2305.5	28.8	581637.9	771850.3	478.9	18944.9	6148.6	27517.3
delta-elevators	3508.0	3164.5	8.0	3508.0	3164.5	22.5	1150836.9	941940.6	577.0	57058.9	10977.7	28732.0
echoMonths	1.0	1.0	1.0	5.0	6.5	1.2	10100.8	2439.4	13.7	315.9	104.9	677.0
electr-len-2	19.0	13.5	50.0	122.0	110.5	28.9	56623.6	7206.1	30.7	1745.2	388.7	14018.8
fishcatch	9.0	5.5	3.0	44.5	38.5	4.6	9240.9	2224.5	18.6	392.8	158.2	1632.0
forestFiresPOR	117.0	106.0	119.0	3.5	1.5	1.1	80509.4	23856.6	65.2	1415.2	457.5	10095.2
fruitfly	1.5	1.5	1.0	1.0	1.0	1.4	8865.0	2513.7	13.5	202.5	117.7	481.4
housing	11.0	16.5	193.0	193.5	179.0	16.2	38822.4	23599.4	48.2	1546.8	709.0	11046.9
hungarian	1.0	12.0	1.0	8.0	7.0	1.5	10668.4	2647.9	23.8	1145.1	237.7	3715.6
kin8nm	3249.0	223.0	1632.0	3181.5	3142.5	30.0	744515.3	214215.1	863.9	43311.0	15021.1	81852.6
laser	38.0	37.5	49.0	219.0	204.5	25.8	52845.0	11177.8	47.1	2445.1	607.1	10876.1
lowbwt	1.5	1.0	1.0	2.5	2.5	1.1	13427.9	3736.2	21.7	662.7	297.5	2024.1
machine-cpu	58.0	6.0	62.0	61.0	53.5	3.8	12929.2	3190.2	14.4	306.1	125.4	915.6
meta	10.0	7.5	10.0	57.5	53.5	8.8	51041.7	13485.9	108.0	1643.8	713.0	61571.9
mortgage	39.0	30.0	99.0	116.5	86.5	24.3	98800.6	9864.1	79.1	2093.8	1277.6	110346.9
pbc	168.5	1.0	1.0	168.5	166.0	1.1	130042.4	9816.0	59.7	1595.8	1450.9	7278.4
pharynx	44.5	8.5	5.0	18.0	17.5	15.4	1765938.6	456127.1	250.9	2635.3	872.4	669277.6
pol	286.5	212.5	1305.0	1281.5	1096.5	30.0	806743.6	258610.5	1832.8	52880.4	23764.5	313991.8
puma8NH	48.5	35.0	35.0	145.0	118.0	23.0	1535174.8	1532370.1	908.6	36663.3	14670.9	48230.5
pwLinear	30.5	74.5	2.0	14.5	16.0	2.5	22179.7	6949.2	18.2	1008.6	218.4	1394.6
quake	3.0	1.0	1.0	5.0	8.0	8.0	193564.5	54371.0	123.5	5729.6	2020.8	5580.9
sensory	218.5	210.5	247.0	220.5	210.5	2.0	127006.5	13150.9	98.0	1739.7	968.4	15085.2
servo	37.5	6.0	37.0	47.5	42.5	3.3	12954.9	3766.1	16.4	536.3	138.4	1744.8
stock	57.5	53.0	253.0	253.5	241.5	29.7	57728.0	31802.6	71.4	8194.5	803.9	25503.8
strike	3.0	9.5	1.0	5.5	12.0	1.1	299610.6	10460.3	67.7	2014.5	1382.2	14200.5
treasury	106.0	87.5	34.0	92.0	87.5	18.0	35993.2	34129.9	73.5	2163.2	741.2	33738.0
triazines	1.0	1.0	77.0	72.0	71.5	3.4	131979.8	15116.5	120.2	1082.6	535.5	14104.1
veteran	17.0	54.0	1.0	48.0	2.5	1.9	49077.1	2723.0	15.5	674.5	104.9	792.2
wankara	503.0	366.0	4.0	417.5	366.0	20.4	178584.5	70115.0	120.6	4344.8	1183.9	19827.2
wisconsin	15.5	17.0	1.0	5.5	4.5	1.0	26232.8	6868.8	65.9	733.5	368.3	5270.9
wizmir	445.5	298.5	2.0	360.0	298.5	14.6	79430.7	19886.3	92.6	3473.3	1317.2	17333.7
#wins		18:31	22:27	37:12	28:21	11:38		2:47	0:49	0:49	0:49	2:47
w-test		0.001	0.236	0.012	0.462	0.000		0.000	0.000	0.000	0.000	0.000

Table A.7: A statistical comparison of the predictive performance of ensembles, for the task of multi-target regression.

	Forests MT :	t-test	Bagg.MT	t-test	MT	t-test	Forests RT	t-test	Bagg.RT	t-test	Bagg.Lol.
class-spec	3.5341		3.7864		6.7591		0.8537		0.8527		2.8580
class-ind	7.0768		8.5728		5.7335		0.8278		0.8262		5.9769
class-fols	4.3792		3.3956		0.9345		0.8950		0.9002		4.0425
Collembola		=		=		=		=		=	
DFlow	0.6763		0.7454		0.7828		0.6041		0.6855		0.7946
DGap	0.6798		0.6948		0.7127		0.7047		0.6973		0.7196
EDM		=		=		=		=		<	
p	0.0152		0.0137		0.0350		0.0175		0.0149		0.0317
aspect	0.9842		0.9962		0.9866		0.9921		1.0061		0.9866
Forestry IRS		=		<		<		=		<	
p	0.0149		0.0138		0.0334		0.0179		0.0148		0.0323
aspect	0.9829		0.9950		0.9962		0.9864		1.0016		0.9953
Forestry SPOT		=		<		<		=		<	
MF0/00 plR	0.6117		0.6109		0.5788		0.5846		0.6268		0.6812
MS0/00 plR	0.5096		0.5484		0.5447		0.5160		0.4575		0.6127
Sigmea-real		=		=		=		=		=	
Dispersal Rate Pollen	173.5425		546.5373		297.7392		0.0412		0.0421		0.0754
Dispersal Seeds	173.2916		379.6341		228.3116		0.0220		0.0219		0.0586
Sigmea-simulated		=		=		=		=		=	
c-class	0.9323		0.9327		0.9523		0.9356		0.9637		0.9570
m-class	0.9171		0.9165		0.9237		0.9116		0.9657		0.9301
x-class	0.9540		0.9485		0.9584		0.9462		1.0199		0.9505
Solar-flare1		=		=		=		<		=	
c-class	0.8652		0.8694		0.8761		0.8755		0.8875		0.8751
m-class	0.9691		0.9707		0.9859		0.9764		0.9968		0.9745
x-class	0.9775		0.9716		0.9711		0.9635		0.9787		0.9905
Solar-flare2		=		=		=		=		=	
Cladophora	0.9231		0.9283		0.9652		0.9315		0.9337		0.9323
Gongrosira	0.9783		0.9796		0.9943		0.9849		0.9839		0.9823
Oedogonium	0.8923		0.8932		0.9047		0.9002		0.8998		0.9029
Stigeoclonium	0.8479		0.8556		0.8729		0.8546		0.8639		0.8817
Melosira	0.9245		0.9313		0.9507		0.9275		0.9345		0.9515
Nitzschia	0.7877		0.7937		0.8235		0.7887		0.7952		0.8172
Audouinella	0.8411		0.8411		0.8664		0.8466		0.8461		0.8552
Erpobdella	0.8967		0.8989		0.9113		0.9001		0.9035		0.9404
Gammarus	0.7828		0.7888		0.8110		0.7933		0.7991		0.8152
Baetis	0.9088		0.9182		0.9386		0.9169		0.9260		0.9304
Hydropsyche	0.9198		0.9245		0.9468		0.9245		0.9322		0.9498
Rhyacophila	0.8507		0.8520		0.8561		0.8485		0.8482		0.8794
Simulium	0.9660		0.9716		0.9918		0.9700		0.9747		0.9848
Tubifex	0.8523		0.8549		0.8765		0.8548		0.8597		0.8871
Water quality		<		<		<		<		<	
t-test			1:0		3:0		3:0		2:0		4:0
w-test			0.001		0.001		0.837		0.102		0.027

Table A.8: A statistical comparison of the model sizes and running times of ensembles, for the task of multi-target regression.

	Model size						Learning time (sec.)					
	Forest.MT:	Bagg.MT	MT	Forest.RT	Bagg.RT	Bagg.Lol.	Forests MT	Bagg.MT	MT	Forests RT	Bagg.RT	Bagg.Lol.
Collemb.	1.1	1.2	1.0	105.5	96.0	1.0	1276587.7	102137.8	626.1	7643.8	3684.5	4412.9
EDM	15.0	4.2	7.0	15.0	9.0	1.8	6684.0	3629.0	16.1	310.2	132.7	88.3
Forest.iRS	584.5	451.0	1.0	584.5	451.0	2.2	502094.9	522805.3	710.2	22309.0	10783.9	2511.3
Forest.SPOT	502.0	430.5	1.0	502.0	430.5	1.5	3075722.3	365028.6	1840.2	39901.2	17867.1	5801.7
Sigmea-real	10.0	3.5	49.0	48.5	47.0	2.8	40074.3	8277.6	19.7	879.6	325.2	282.6
Sigmea-sim.	33.5	36.5	30.0	33.5	36.5	26.7	236026.9	67697.1	297.8	17516.1	8995.6	8718.1
Solar-flare1	1.0	1.3	12.0	6.5	1.0	1.0	50864.8	16071.0	21.9	659.1	320.6	444.5
Solar-flare2	13.0	13.0	13.0	13.0	13.0	1.0	28540.6	9463.2	95.2	1548.7	691.1	1211.7
Water quality	64.5	47.5	53.0	64.5	47.5	3.1	330572.0	224589.1	1005.7	22288.1	8924.7	1483.9
#wins		4:5	4:5	9:0	5:4	2:7		1:8	0:9	0:9	0:9	0:9
w-test		0.078	0.375	0.250	0.688	0.016		0.020	0.004	0.004	0.004	0.004

Table A.9: A statistical comparison of the predictive performance of forests of M5' model trees with a different number of trees for the task of single-target regression.

	RF(100 MT)	t-test	RF(50 MT)	t-test	RF(25 MT)
abalone	0.6535	=	0.6533	=	0.6531
analcat	0.1474	=	0.1474	=	0.1454
auto93	0.4279	=	0.4272	=	0.4313
autoHorse	0.2877	=	0.2909	<	0.3156
autoMpg	0.3286	=	0.3335	=	0.3314
auto-price	0.4101	=	0.4146	=	0.4174
bank8FM	0.1895	=	0.1898	=	0.1920
baseball	0.5496	=	0.5614	=	0.5492
baskball	0.8035	>	0.7948	=	0.8006
bodyfat	0.1591	=	0.1598	=	0.1679
breastTumor	0.9558	=	0.9546	=	0.9544
cal-housing	0.4236	=	0.4252	=	0.4298
cholesterol	1.0026	=	0.9838	=	1.0068
cleveland	0.6925	=	0.7035	=	0.7013
cloud	0.3953	=	0.3919	=	0.4021
concrete	0.3006	=	0.3046	=	0.3200
cpu	0.1613	=	0.1544	=	0.2202
cpu-act	0.1265	=	0.1269	=	0.1284
dailyElectrEner	0.4065	=	0.4166	=	0.4114
delta-ailerons	0.5266	=	0.5274	=	0.5281
delta-elevators	0.6734	=	0.6739	=	0.6795
echoMonths	0.7189	=	0.7201	=	0.7213
electr-len-2	0.0398	=	0.0468	=	0.0414
fishcatch	0.1362	=	0.1388	=	0.1400
forestFiresPOR	1.0080	=	1.0190	=	1.0032
fruitfly	1.0216	=	1.0177	=	1.0290
housing	0.3648	=	0.3465	=	0.3601
hungarian	0.7195	=	0.7285	=	0.7273
kin8nm	0.6995	=	0.5758	=	0.6036
laser	0.1328	=	0.1366	=	0.1401
lowbwt	0.6178	=	0.6369	=	0.6353
machine-cpu	0.2863	=	0.3057	=	0.2897
meta	0.6285	=	5991.7710	=	12560.1648
mortgage	0.0221	=	0.0226	=	0.0240
pbc	0.8042	=	0.8081	=	0.8106
pharynx	0.2974	=	0.3007	=	0.3040
puma8NH	0.5638	=	0.5636	<	0.5657
pwLinear	0.3441	=	0.3240	=	0.3304
quake	1.0232	=	1.0342	=	1.1682
sensory	0.8237	=	0.8213	=	0.8241
servo	0.2721	=	0.2616	=	0.2782
stock	0.1049	=	0.1051	=	0.1058
strike	0.9067	=	0.9074	=	0.9110
treasury	0.0534	=	0.0536	=	0.0548
triazines	0.8049	=	0.8136	=	0.7949
veteran	0.9493	=	0.9467	=	0.9411
wankara	0.0766	=	0.0770	=	0.0773
wisconsin	0.9488	=	0.9492	=	0.9538
wizmir	0.0765	=	0.0765	=	0.0770
t-test			0:1		2:0
w-test			0.064		0.000

Table A.10: A statistical comparison of the model sizes and running times of forests of M5' model trees for the task of single-target regression.

	Model size			Learning time (sec.)		
	RF(100 MT)	RF(50 MT)	RF(25 MT)	RF(100 MT)	RF(50 MT)	RF(25 MT)
abalone	23.5	13.5	19.0	327840.8	300031.6	90246.9
analcat	55.5	43.5	16.0	194682.0	24286.5	12853.4
auto93	1.0	5.0	1.5	13216.0	7221.9	4346.6
autoHorse	10.0	49.5	56.0	83234.8	43382.4	21305.2
autoMpg	155.0	155.5	161.0	31505.0	16491.3	27920.1
auto-price	8.5	4.0	7.5	11717.0	6475.1	6058.5
bank8FM	2377.0	2437.5	2365.0	2456439.2	324667.2	304354.7
baseball	2.5	3.5	116.5	51681.8	12248.0	20916.0
baskball	1.0	1.0	1.0	8457.8	4375.8	2463.6
bodyfat	63.0	72.0	40.0	32968.0	8898.1	6970.7
breastTumor	1.5	1.0	3.0	199158.5	47568.9	7447.6
cal-housing	7278.5	7090.0	7189.5	1749791.5	871515.5	495852.7
cholesterol	1.0	1.0	1.0	24521.2	24951.2	14836.1
cleveland	1.0	88.5	89.0	16164.2	9124.3	10268.4
cloud	2.5	1.0	2.0	8873.0	4530.4	2609.4
concrete	32.0	34.5	37.0	378604.4	40559.7	29789.9
cpu	17.0	9.0	11.0	11454.6	27391.5	7197.0
cpu-act	1927.5	1896.0	1917.0	680543.6	366597.0	176832.3
dailyElectrEner	74.5	80.5	148.5	27731.8	16716.9	8221.8
delta-ailerons	2522.0	2515.0	2506.5	581637.9	317841.6	571575.4
delta-elevators	3508.0	3526.0	3491.5	1150836.9	367030.5	189756.4
echoMonths	1.0	1.0	1.0	10100.8	4992.0	2953.3
electr-len-2	19.0	20.5	13.5	56623.6	86443.9	10814.6
fishcatch	9.0	7.0	7.5	9240.9	5027.8	2783.3
forestFiresPOR	117.0	122.5	11.0	80509.4	14783.4	8823.6
fruitfly	1.5	1.0	1.0	8865.0	4898.6	2612.7
housing	11.0	12.5	13.5	38822.4	20329.5	18071.3
hungarian	1.0	22.5	13.0	10668.4	5804.6	3325.3
kin8nm	3249.0	195.0	3269.0	744515.3	2231298.8	345005.7
laser	38.0	24.5	35.5	52845.0	50555.3	17919.0
lowbwt	1.5	1.5	2.0	13427.9	50708.6	4486.7
machine-cpu	58.0	6.5	4.0	12929.2	6229.5	3498.7
meta	10.0	37.5	37.5	51041.7	28024.2	38594.6
mortgage	39.0	132.5	121.0	98800.6	180214.4	11391.1
pbc	168.5	165.5	172.0	130042.4	19655.0	11108.4
pharynx	44.5	50.5	44.0	1765938.6	57289.0	392165.5
puma8NH	48.5	42.0	48.5	1535174.8	2124603.2	210734.8
pwLinear	30.5	77.0	78.5	22179.7	8668.3	4634.6
quake	3.0	1.0	4.0	193564.5	100668.6	46251.2
sensory	218.5	223.5	221.5	127006.5	26247.5	14620.6
servo	37.5	37.5	32.0	12954.9	5443.3	3053.1
stock	57.5	37.5	49.0	57728.0	29893.7	16157.1
strike	3.0	4.0	8.0	299610.6	21544.2	11947.4
treasury	106.0	110.0	98.5	35993.2	79147.7	11631.9
triazines	1.0	1.0	1.0	131979.8	27585.1	16244.6
veteran	17.0	53.5	54.0	49077.1	5506.0	3068.3
wankara	503.0	414.0	442.5	178584.5	54083.2	28107.4
wisconsin	15.5	16.0	1.0	26232.8	14409.0	7398.8
wizmir	445.5	424.5	444.5	79430.7	40938.4	22529.0
#wins		29:20	25:24		8:41	0:49
w-test		0.885	0.770		0.000	0.000

Table A.11: Comparing the performance of M5' to Lolimot.

	OSA		**SIM**		**numLM**	**time**	
	M5'	**Lolimot**$_S$	**M5'**	**Lolimot**$_S$		**M5'**	**Lolimot**$_S$
CSTR (Ca)	0.0151	0.0035	0.0151	0.0091	12.0000	0.5100	42.8500
CSTR' (Ca)	0.0155	0.0154	0.0167	0.0171	4.0000	1.1190	5.4250
CSTR (T)	0.0040	0.0023	0.0125	0.0047	3.0000	0.5140	2.6920
CSTR' (T)	0.0242	0.0173	0.0261	0.0194	4.0000	1.0170	4.9090
GLS (h_1)	0.0050	0.0046	0.1132	0.0646	2.0000	0.0910	0.0370
GLS (p_1)	0.1096	0.1066	0.1881	0.1764	4.0000	0.0670	0.1080
Narendra	0.0591	0.0249	0.2098	0.0441	41.0000	0.1280	85.8620
Narendra'	0.0750	0.0724	0.1456	0.1286	10.0000	0.3710	5.9220
pH$_B$	0.1350	0.0717	0.2937	0.2021	8.0000	0.1450	0.2780
pH'$_B$	0.0934	0.0849	0.2426	0.2335	2.0000	0.0540	0.0140
pH$_A$	0.0553	0.0657	0.1267	0.1296	6.0000	0.0200	0.1080
pH'$_A$	0.1125	0.1328	0.6739	0.2868	3.0000	0.0310	0.0230
SteamGen (y_1)	0.0220	0.0191	0.2287	0.1446	2.0000	1.2100	2.8280
SteamGen (y_2)	0.2685	0.2858	0.2976	0.3092	48.0000	1.7920	970.3870
SteamGen (y_3)	0.0906	0.0620	0.2055	0.1427	2.0000	1.9130	2.4270
SteamGen (y_4)	0.0493	0.0489	0.1579	0.1392	2.0000	1.5480	2.3970
Robot (τ_1)	0.0922	0.0905	0.1639	0.1275	3.0000	0.6230	0.9170
Robot (τ_2)	0.1029	0.0967	NaN	48.1917	2.0000	0.4910	0.4590
Robot (τ_3)	0.0738	0.0690	0.2313	0.1830	2.0000	0.4470	0.5770
Robot (τ_4)	0.0684	0.0620	NaN	0.1043	2.0000	0.6470	0.5700
Robot (τ_5)	0.1836	0.1742	0.8780	0.3031	2.0000	0.5690	0.5730
Robot (τ_6)	0.0786	0.0751	0.1456	0.1205	2.0000	0.5910	0.5750
Robot (τ_7)	0.0629	0.0610	0.0858	0.0770	2.0000	0.7000	0.6700
Winding (T_1)	0.3220	0.2890	0.4010	0.3384	4.0000	0.8000	2.7740
Winding (T_2)	0.1071	0.0749	0.2788	0.1487	31.0000	0.6480	96.7630

Table A.12: Comparing M5' to a version with fuzzy/soft estimation, and to Lolimot of the same size.

	OSA			SIM			numLM
	M5'	$M5'_{SOFT}$	**Lolimot**$_S$	**M5'**	$M5'_{SOFT}$	**Lolimot**$_S$	
CSTR (Ca)	0.0151	0.0016	0.0035	0.0151	0.0026	0.0091	12.0000
CSTR' (Ca)	0.0155	0.0163	0.0154	0.0167	0.0189	0.0171	4.0000
CSTR (T)	0.0040	0.0023	0.0023	0.0125	0.0051	0.0047	3.0000
CSTR' (T)	0.0242	0.0213	0.0173	0.0261	0.0233	0.0194	4.0000
GLS (h_1)	0.0050	0.0046	0.0046	0.1132	0.0666	0.0646	2.0000
GLS (p_1)	0.1096	0.1120	0.1066	0.1881	0.2003	0.1764	4.0000
Narendra	0.0591	0.0354	0.0249	0.2098	0.1106	0.0441	41.0000
Narendra'	0.0750	0.0751	0.0724	0.1456	0.1510	0.1286	10.0000
pH$_B$	0.1350	0.0732	0.0717	0.2937	0.2251	0.2021	8.0000
pH'$_B$	0.0934	0.1009	0.0849	0.2426	0.2747	0.2335	2.0000
pH$_A$	0.0553	0.0595	0.0657	0.1267	0.1548	0.1296	6.0000
pH'$_A$	0.1125	0.1346	0.1328	0.6739	0.2663	0.2868	3.0000
SteamGen (y_1)	0.0220	0.0191	0.0191	0.2287	0.1470	0.1446	2.0000
SteamGen (y_2)	0.2685	0.2233	0.2858	0.2976	0.2512	0.3092	48.0000
SteamGen (y_3)	0.0906	0.0627	0.0620	0.2055	0.1445	0.1427	2.0000
SteamGen (y_4)	0.0493	0.0489	0.0489	0.1579	0.1538	0.1392	2.0000
Robot (τ_1)	0.0922	0.0888	0.0905	0.1639	0.1522	0.1275	3.0000
Robot (τ_2)	0.1029	0.1027	0.0967	NaN	NaN	48.1917	2.0000
Robot (τ_3)	0.0738	0.0707	0.0690	0.2313	0.2340	0.1830	2.0000
Robot (τ_4)	0.0684	0.0671	0.0620	NaN	0.2486	0.1043	2.0000
Robot (τ_5)	0.1836	0.1837	0.1742	0.8780	0.4004	0.3031	2.0000
Robot (τ_6)	0.0786	0.0781	0.0751	0.1456	0.1499	0.1205	2.0000
Robot (τ_7)	0.0629	0.0618	0.0610	0.0858	0.0825	0.0770	2.0000
Winding (T_1)	0.3220	0.2871	0.2890	0.4010	0.3449	0.3384	4.0000
Winding (T_2)	0.1071	0.0816	0.0749	0.2788	0.3260	0.1487	31.0000

Table A.13: Evaluation results of the M5' smoothing variants.

	OSA			SIM			numLM		
	M5'	**M5'**$_{BSM}$	**M5'**$_{Fuzz}$	**M5'**	**M5'**$_{BSM}$	**M5'**$_{Fuzz}$	**M5'**	**M5'**$_{BSM}$	**M5'**$_{Fuzz}$
CSTR (Ca)	0.0151	0.0147	0.0150	0.0151	0.0147	0.0150	0.5700	0.2530	2.2620
CSTR' (Ca)	0.0155	0.0152	0.0152	0.0167	0.0164	0.0165	0.6750	0.6120	5.7790
CSTR (T)	0.0040	0.0040	0.0039	0.0125	0.0119	0.0124	0.2790	0.2810	2.7860
CSTR' (T)	0.0242	0.0239	0.0241	0.0261	0.0258	0.0260	0.5870	0.5830	5.7610
GLS (h_1)	0.0050	0.0049	0.0050	0.1132	0.1107	0.1124	0.0110	0.0110	0.1450
GLS (p_1)	0.1096	0.1099	0.1090	0.1881	0.1895	0.1874	0.0250	0.0250	0.2580
Narendra	0.0591	0.0474	0.0565	0.2098	0.1112	0.1048	0.0460	0.0470	0.5340
Narendra'	0.0750	0.0713	0.0844	0.1456	0.1318	0.1534	0.1140	0.1220	1.1720
pH$_B$	0.1350	0.0920	0.1089	0.2937	0.2509	0.2589	0.0160	0.0160	0.1770
pH'$_B$	0.0934	0.0940	0.0922	0.2426	0.2460	0.2427	0.0200	0.0200	0.1980
pH$_A$	0.0553	0.0585	0.0602	0.1267	0.1241	0.1758	0.0080	0.0080	0.0980
pH'$_A$	0.1125	0.1125	0.1120	0.6739	0.3676	0.3790	0.0150	0.0150	0.1470
SteamGen (y_1)	0.0220	0.0219	0.0219	0.2287	0.2270	0.2276	0.5550	0.5620	5.1520
SteamGen (y_2)	0.2685	0.2303	0.2366	0.2976	0.2516	0.2754	0.9160	0.9120	7.5880
SteamGen (y_3)	0.0906	0.0901	0.0905	0.2055	0.2041	0.2054	1.1850	1.1740	11.7430
SteamGen (y_4)	0.0493	0.0493	0.0493	0.1579	0.1579	0.1579	0.9370	0.9370	8.4620
Robot (τ_1)	0.0922	0.0917	0.0906	0.1639	0.2344	0.1966	0.1910	0.1920	1.8780
Robot (τ_2)	0.1029	0.1027	0.1027	NaN	NaN	NaN	0.2030	0.2030	1.9230
Robot (τ_3)	0.0738	0.0736	0.0735	0.2313	0.2307	0.2303	0.1870	0.1900	1.9250
Robot (τ_4)	0.0684	0.0683	0.0682	NaN	NaN	NaN	0.1850	0.1910	1.9440
Robot (τ_5)	0.1836	0.1834	0.1830	0.8780	0.8360	0.8816	0.2300	0.2370	2.2540
Robot (τ_6)	0.0786	0.0785	0.0783	0.1456	0.1451	0.1450	0.1830	0.1870	1.8510
Robot (τ_7)	0.0629	0.0628	0.0629	0.0858	0.0857	0.0849	0.1820	0.1840	1.8940
Winding (T_1)	0.3220	0.3187	0.3254	0.4010	0.3804	0.4379	0.2930	0.2920	2.7970
Winding (T_2)	0.1071	0.0973	0.1018	0.2788	0.2167	0.2193	0.2150	0.2160	2.1760

Table A.14: Results for a Lolimot modification which evaluates candidate splits using a different procedure. The reported running times are those required for building the model tree.

	OSA		**SIM**		**numLM**	**time**	
	Lolimot	**Lolimot**$_{ME}$	**Lolimot**	**Lolimot**$_{ME}$		**Lolimot**	**Lolimot**$_{ME}$
CSTR (Ca)	0.0034	0.0018	0.0073	0.0034	30.0000	435.1640	18.3560
CSTR' (Ca)	0.0143	0.0169	0.0159	0.0184	12.0000	97.0430	5.7570
CSTR (T)	0.0020	0.0023	0.0036	0.0051	25.0000	364.2050	13.3370
CSTR' (T)	0.0113	0.0141	0.0120	0.0151	8.0000	26.2120	1.6650
GLS (h_1)	0.0046	0.0046	0.0646	0.0646	2.0000	0.0490	0.0260
GLS (p_1)	0.0999	0.1063	0.1633	0.1754	7.0000	1.4350	0.3760
Narendra	0.0505	0.0468	0.1036	0.0948	14.0000	11.4990	1.5920
Narendra'	0.0704	0.0759	0.1242	0.1400	8.0000	3.1490	0.4480
pH$_B$	0.0594	0.0675	0.1459	0.1988	12.0000	0.7660	0.3400
pH'$_B$	0.0984	0.0850	0.2836	0.2265	12.0000	0.8260	0.6520
pH$_A$	0.0657	0.0657	0.1123	0.1123	9.0000	0.5430	0.4570
pH'$_A$	0.1059	0.0999	0.1399	0.1232	30.0000	2.9640	3.0170
SteamGen (y_1)	0.0191	0.0192	0.1446	0.1449	2.0000	5.5600	2.2580
SteamGen (y_2)	0.2303	0.2325	0.2732	0.2950	5.0000	33.6790	13.0040
SteamGen (y_3)	0.0627	0.0627	0.1440	0.1440	1.0000	0.0840	0.0570
SteamGen (y_4)	0.0467	0.0488	0.1265	0.1535	5.0000	22.3080	8.4400
Robot (τ_1)	0.0774	0.0768	0.0840	0.0827	8.0000	10.1160	3.9910
Robot (τ_2)	0.0775	0.0791	0.1523	0.1933	28.0000	94.1000	31.5620
Robot (τ_3)	0.0751	0.0727	0.1029	0.1100	28.0000	59.1450	18.3340
Robot (τ_4)	0.0542	0.0558	0.0593	0.0615	5.0000	2.6970	1.2010
Robot (τ_5)	0.1638	0.1622	0.2421	0.2348	5.0000	3.7550	1.7650
Robot (τ_6)	0.0712	0.0714	0.0962	0.0978	6.0000	6.4620	2.7250
Robot (τ_7)	0.0565	0.0568	0.0646	0.0675	4.0000	2.3380	1.0500
Winding (T_1)	0.2837	0.2777	0.3359	0.3287	7.0000	6.0890	1.5860
Winding (T_2)	0.0813	0.0813	0.1893	0.1893	3.0000	2.6480	0.6580

Table A.15: Evaluating Lolimot - several split cut-points. Prediction and output error results.

	OSA				**SIM**			
	Lolimot$_{ME}$	**Lolimot**$_{C2}$	**Lolimot**$_{C4}$	**Lolimot**$_{C8}$	**Lolimot**$_{ME}$	**Lolimot**$_{C2}$	**Lolimot**$_{C4}$	**Lolimot**$_{C8}$
CSTR (Ca)	0.0018	0.0039	0.0026	0.0035	0.0034	0.0079	0.0063	0.0087
CSTR' (Ca)	0.0169	0.0216	0.0196	0.0204	0.0184	0.0229	0.0219	0.0229
CSTR (T)	0.0023	0.0023	0.0024	0.0026	0.0051	0.0046	0.0050	0.0049
CSTR' (T)	0.0141	0.0116	0.0109	0.0185	0.0151	0.0126	0.0116	0.0202
GLS (h_1)	0.0046	0.0046	0.0046	0.0046	0.0646	0.0852	0.0850	0.0850
GLS (p_1)	0.1063	0.1038	0.1095	0.1134	0.1754	0.1695	0.1869	0.2259
Narendra	0.0468	0.0332	0.0367	0.0363	0.0948	0.0768	0.0767	0.0794
Narendra'	0.0759	0.0803	0.0544	0.0704	0.1400	0.1345	0.1031	0.1306
pH$_B$	0.0675	0.0704	0.0701	0.0723	0.1988	0.2001	0.2084	0.2115
pH'$_B$	0.0850	0.1038	0.1092	0.1022	0.2265	0.2540	0.2631	0.2530
pH$_A$	0.0657	0.0676	0.0512	0.0560	0.1123	0.1282	0.0899	0.1307
pH'$_A$	0.0999	0.0963	0.0906	0.0870	0.1232	0.1227	0.1129	0.1237
SteamGen (y_1)	0.0192	0.0192	0.0208	0.0195	0.1449	0.1524	0.1632	0.1575
SteamGen (y_2)	0.2325	0.2281	0.2374	0.2354	0.2950	0.2924	0.3190	0.3144
SteamGen (y_3)	0.0627	0.0629	0.0627	0.0629	0.1440	0.1442	0.1440	0.1412
SteamGen (y_4)	0.0488	0.0488	0.0488	0.0487	0.1535	0.1496	0.1431	0.1430
Robot (τ_1)	0.0768	0.0764	0.0786	0.0767	0.0827	0.0836	0.0916	0.0770
Robot (τ_2)	0.0791	0.0923	0.0816	0.0819	0.1933	0.1706	0.1527	0.1970
Robot (τ_3)	0.0727	0.0824	0.0627	0.0635	0.1100	0.1391	0.0954	0.1055
Robot (τ_4)	0.0558	0.0534	0.0544	0.0563	0.0615	0.0571	0.0637	0.0718
Robot (τ_5)	0.1622	0.1668	0.1702	0.1714	0.2348	0.2389	0.2747	0.2127
Robot (τ_6)	0.0714	0.0711	0.0722	0.0741	0.0978	0.0999	0.0990	0.1077
Robot (τ_7)	0.0568	0.0567	0.0548	0.0583	0.0675	0.0740	0.0648	0.0774
Winding (T_1)	0.2777	0.2844	0.2701	0.2780	0.3287	0.3372	0.3366	0.3330
Winding (T_2)	0.0813	0.0874	0.0814	0.0810	0.1893	0.1732	0.1821	0.1843

Table A.16: Evaluating Lolimot - several split cut-points. Model sizes and learning times (seconds) are shown. The reported running times are those required for tuning of the parameters and building the model tree.

	numLM				**time**			
	Lolimot$_{ME}$	**Lolimot**$_{C2}$	**Lolimot**$_{C4}$	**Lolimot**$_{C8}$	**Lolimot**$_{ME}$	**Lolimot**$_{C2}$	**Lolimot**$_{C4}$	**Lolimot**$_{C8}$
CSTR (Ca)	30.0	30.0	30.0	30.0	301.1	1258.2	2275.8	4388.8
CSTR' (Ca)	12.0	30.0	10.0	10.0	217.3	886.2	1516.4	3020.9
CSTR (T)	25.0	30.0	24.0	28.0	246.2	1164.1	2201.0	4470.9
CSTR' (T)	8.0	7.0	7.0	6.0	190.2	765.2	1499.6	2993.8
GLS (h_1)	2.0	2.0	1.0	1.0	5.8	15.9	31.5	53.7
GLS (p_1)	7.0	28.0	4.0	11.0	5.7	23.1	29.7	61.6
Narendra	14.0	30.0	30.0	30.0	28.5	114.1	215.4	418.7
Narendra'	8.0	10.0	21.0	8.0	25.2	87.1	154.5	279.9
pH$_B$	12.0	6.0	7.0	6.0	3.3	6.8	11.4	20.6
pH'$_B$	12.0	12.0	10.0	7.0	3.7	7.1	11.7	21.3
pH$_A$	9.0	7.0	21.0	27.0	3.0	5.6	15.9	39.2
pH'$_A$	30.0	30.0	21.0	30.0	5.9	13.1	16.1	41.4
SteamGen (y_1)	2.0	3.0	28.0	16.0	355.7	1810.5	4739.8	7153.2
SteamGen (y_2)	5.0	8.0	6.0	6.0	318.6	1834.5	3582.6	6562.9
SteamGen (y_3)	1.0	2.0	1.0	4.0	352.6	1802.3	3488.0	6476.2
SteamGen (y_4)	5.0	3.0	4.0	4.0	319.0	1760.1	3525.3	6499.8
Robot (τ_1)	8.0	8.0	6.0	13.0	43.2	236.9	455.6	993.0
Robot (τ_2)	28.0	20.0	16.0	19.0	58.3	264.0	486.0	982.8
Robot (τ_3)	28.0	20.0	12.0	14.0	62.4	286.1	494.7	981.9
Robot (τ_4)	5.0	9.0	5.0	4.0	40.4	218.2	410.2	811.0
Robot (τ_5)	5.0	7.0	3.0	13.0	44.2	232.0	440.5	987.6
Robot (τ_6)	6.0	4.0	5.0	3.0	41.1	205.3	409.1	799.1
Robot (τ_7)	4.0	3.0	6.0	2.0	43.9	223.8	455.8	883.1
Winding (T_1)	7.0	3.0	10.0	5.0	49.2	222.2	464.7	886.4
Winding (T_2)	3.0	21.0	3.0	3.0	46.5	274.8	441.4	878.8

Table A.17: Evaluating Lolimot - optimizing overlaps. The reported running times are those required for tuning of the parameters and building the model tree. The running times of Lolimot$_{ME}$ are similar, but not identical, to those reported in Table A.16.

	OSA		**SIM**		**numLM**		**time**	
	Lolimot$_{ME}$	**Lolimot**$_{ksig}$	**Lolimot**$_{ME}$	**Lolimot**$_{ksig}$	**Lolimot**$_{ME}$	**Lolimot**$_{ksig}$	**Lolimot**$_{ME}$	**Lolimot**$_{ksig}$
CSTR (Ca)	0.0018	0.0018	0.0034	0.0031	30.0000	30.0000	286.7600	241.3890
CSTR' (Ca)	0.0169	0.0177	0.0184	0.0195	12.0000	14.0000	207.0690	171.5370
CSTR (T)	0.0023	0.0021	0.0051	0.0039	25.0000	27.0000	273.3760	240.2040
CSTR' (T)	0.0141	0.0113	0.0151	0.0120	8.0000	10.0000	203.3590	167.4070
GLS (h_1)	0.0046	0.0045	0.0646	0.1139	2.0000	12.0000	6.4880	64.1190
GLS (p_1)	0.1063	0.1036	0.1754	0.1693	7.0000	16.0000	6.9540	60.8890
Narendra	0.0468	0.0342	0.0948	0.0680	14.0000	30.0000	34.7960	88.1650
Narendra'	0.0759	0.0693	0.1400	0.1235	8.0000	12.0000	25.3800	60.6140
pH$_B$	0.0675	0.0678	0.1988	0.1175	12.0000	18.0000	4.0520	22.2290
pH'$_B$	0.0850	0.0770	0.2265	0.2108	12.0000	12.0000	3.4720	21.7100
pH$_A$	0.0657	0.0578	0.1123	0.0990	9.0000	29.0000	4.6400	32.6940
pH'$_A$	0.0999	0.0960	0.1232	0.1183	30.0000	29.0000	4.9510	27.5160
SteamGen (y_1)	0.0192	0.0205	0.1449	0.1829	2.0000	28.0000	366.6350	1255.8660
SteamGen (y_2)	0.2325	0.2320	0.2950	0.2739	5.0000	30.0000	369.2010	1259.1290
SteamGen (y_3)	0.0627	0.0627	0.1440	0.1440	1.0000	10.0000	284.5960	1055.5620
SteamGen (y_4)	0.0488	0.0499	0.1535	0.1540	5.0000	1.0000	275.6080	993.6560
Robot (τ_1)	0.0768	0.0750	0.0827	0.0797	8.0000	11.0000	36.4080	207.5720
Robot (τ_2)	0.0791	0.0812	0.1933	0.1733	28.0000	13.0000	65.1310	248.8330
Robot (τ_3)	0.0727	0.0654	0.1100	0.1303	28.0000	6.0000	66.4210	245.8590
Robot (τ_4)	0.0558	0.0583	0.0615	0.0605	5.0000	18.0000	46.5190	262.4740
Robot (τ_5)	0.1622	0.1594	0.2348	0.2142	5.0000	5.0000	46.4230	245.9110
Robot (τ_6)	0.0714	0.0730	0.0978	0.0971	6.0000	8.0000	35.3030	201.2070
Robot (τ_7)	0.0568	0.0571	0.0675	0.0670	4.0000	8.0000	34.8410	201.0010
Winding (T_1)	0.2777	0.2749	0.3287	0.3251	7.0000	14.0000	57.3940	180.8080
Winding (T_2)	0.0813	0.0864	0.1893	0.1729	3.0000	28.0000	48.6080	191.2200

Table A.18: Evaluating Lolimot - using output error or prediction error while learning. In both algorithm variants the trees are of equal size.

	OSA		**SIM**		**numLM**	**time**	
	Lolimot$_{ksig}$	**Lolimot**$_{osa}$	**Lolimot**$_{ksig}$	**Lolimot**$_{osa}$		**Lolimot**$_{ksig}$	**Lolimot**$_{osa}$
CSTR (Ca)	0.0018	0.0020	0.0031	0.0032	30.0000	241.3890	241.9190
CSTR' (Ca)	0.0177	0.0165	0.0195	0.0177	14.0000	171.5370	171.5230
CSTR (T)	0.0021	0.0024	0.0039	0.0044	27.0000	240.2040	240.1680
CSTR' (T)	0.0113	0.0103	0.0120	0.0110	10.0000	167.4070	167.3760
GLS (h_1)	0.0045	0.0040	0.1139	0.4826	12.0000	64.1190	64.1070
GLS (p_1)	0.1036	0.1066	0.1693	0.1824	16.0000	60.8890	61.2430
Narendra	0.0342	0.0345	0.0680	0.0769	30.0000	88.1650	88.5960
Narendra'	0.0693	0.0683	0.1235	0.1213	12.0000	60.6140	60.7620
pH$_B$	0.0678	0.0525	0.1175	0.1288	18.0000	22.2290	22.6040
pH'$_B$	0.0770	0.0777	0.2108	0.2153	12.0000	21.7100	21.5220
pH$_A$	0.0578	0.0405	0.0990	0.0759	29.0000	32.6940	33.0820
pH'$_A$	0.0960	0.1070	0.1183	0.1531	29.0000	27.5160	27.9650
SteamGen (y_1)	0.0205	0.0211	0.1829	0.1682	28.0000	1255.8660	1256.4640
SteamGen (y_2)	0.2320	0.2373	0.2739	0.3019	30.0000	1259.1290	1259.7800
SteamGen (y_3)	0.0627	0.0627	0.1440	0.1436	10.0000	1055.5620	1055.8670
SteamGen (y_4)	0.0499	0.0499	0.1540	0.1540	1.0000	993.6560	993.6510
Robot (τ_1)	0.0750	0.0745	0.0797	0.0774	11.0000	207.5720	207.5720
Robot (τ_2)	0.0812	0.0921	0.1733	0.1624	13.0000	248.8330	248.7610
Robot (τ_3)	0.0654	0.0617	0.1303	0.1182	6.0000	245.8590	245.8670
Robot (τ_4)	0.0583	0.0620	0.0605	0.0624	18.0000	262.4740	262.7770
Robot (τ_5)	0.1594	0.1581	0.2142	0.2237	5.0000	245.9110	245.9120
Robot (τ_6)	0.0730	0.0813	0.0971	0.1057	8.0000	201.2070	201.2410
Robot (τ_7)	0.0571	0.0533	0.0670	0.0690	8.0000	201.0010	200.9840
Winding (T_1)	0.2749	0.2898	0.3251	0.3279	14.0000	180.8080	180.8580
Winding (T_2)	0.0864	0.1123	0.1729	0.2998	28.0000	191.2200	191.2020

Table A.19: Evaluating Lolimot and L++ for multi-output modeling.

	OSA		**SIM**		**numLM**		**time**	
	Lolimot$_{MO}$	**L++**$_{MO}$	**Lolimot**$_{MO}$	**L++**$_{MO}$	**Lolimot**$_{MO}$	**L++**$_{MO}$	**Lolimot**$_{MO}$	**L++**$_{MO}$
CSTR (Ca)	0.0025	0.0019	0.0372	0.0166	27.0000	30.0000	278.3130	18.6060
CSTR (T)	0.0040	0.0022	0.0279	0.0134	27.0000	30.0000	278.3130	18.6060
CSTR' (Ca)	0.0236	0.0133	0.0332	0.0191	29.0000	17.0000	441.8160	7.3520
CSTR' (T)	0.0241	0.0136	0.0308	0.0200	29.0000	17.0000	441.8160	7.3520
GLS (p_1)	0.0955	0.0923	0.2315	0.2370	5.0000	30.0000	0.6000	4.1950
GLS (h_1)	0.0044	0.0040	0.1496	0.1462	5.0000	30.0000	0.6000	4.1950
SteamGen (y_1)	0.0192	0.0194	0.1416	0.1573	9.0000	21.0000	76.2570	117.2630
SteamGen (y_2)	0.2278	0.2373	0.2611	0.2878	9.0000	21.0000	76.2570	117.2630
SteamGen (y_3)	0.0646	0.0643	0.2031	0.1999	9.0000	21.0000	76.2570	117.2630
SteamGen (y_4)	0.0472	0.0468	0.1295	0.1365	9.0000	21.0000	76.2570	117.2630
Robot (τ_1)	0.0820	0.0879	0.1045	0.0995	19.0000	14.0000	64.3160	19.8540
Robot (τ_2)	0.0847	0.0918	0.2333	0.1745	19.0000	14.0000	64.3160	19.8540
Robot (τ_3)	0.0680	0.0768	0.1463	0.1102	19.0000	14.0000	64.3160	19.8540
Robot (τ_4)	0.0569	0.0618	0.0723	0.0812	19.0000	14.0000	64.3160	19.8540
Robot (τ_5)	0.1661	0.1928	0.2497	0.2301	19.0000	14.0000	64.3160	19.8540
Robot (τ_6)	0.0864	0.0734	0.1308	0.0996	19.0000	14.0000	64.3160	19.8540
Robot (τ_7)	0.0582	0.0575	0.0738	0.0777	19.0000	14.0000	64.3160	19.8540
Winding (T_1)	0.2816	0.2823	0.3534	0.3687	6.0000	19.0000	5.9130	10.9080
Winding (T_2)	0.0806	0.1045	0.1869	0.2149	6.0000	19.0000	5.9130	10.9080

Table A.20: Comparing the single-output performance of Lolimot trees and Bagging of Lolimot trees.

	OSA		SIM		time	
	Lolimot	**Bagg.Lolimot**	**Lolimot**	**Bagg.Lolimot**	**Lolimot**	**Bagg.Lolimot**
CSTR (Ca)	0.0034±0.0000	0.0024±0.0000	0.0073±0.0000	0.0049±0.0001	509.3090	240992.1532
CSTR' (Ca)	0.0143±0.0000	0.0135±0.0002	0.0159±0.0000	0.0152±0.0002	55.6940	34955.9190
CSTR (T)	0.0020±0.0000	0.0020±0.0000	0.0036±0.0000	0.0035±0.0000	397.4090	151048.7654
CSTR' (T)	0.0113±0.0000	0.0106±0.0003	0.0120±0.0000	0.0113±0.0003	25.6930	25227.9896
GLS (h_1)	0.0046±0.0000	0.0046±0.0000	0.0646±0.0000	0.0664±0.0012	0.1000	16.1046
GLS (p_1)	0.0999±0.0000	0.0986±0.0003	0.1633±0.0000	0.1574±0.0009	1.0240	190.4570
Narendra	0.0505±0.0000	0.0493±0.0002	0.1036±0.0000	0.0976±0.0008	9.1530	3420.2020
Narendra'	0.0704±0.0000	0.0701±0.0001	0.1242±0.0000	0.1240±0.0001	3.6890	2367.7390
pH$_B$	0.0594±0.0000	0.0577±0.0003	0.1459±0.0000	0.1404±0.0023	0.6690	275.4248
pH'$_B$	0.0984±0.0000	0.0893±0.0017	0.2836±0.0000	0.2512±0.0054	0.4980	167.1672
pH$_A$	0.0657±0.0000	0.0638±0.0003	0.1123±0.0000	0.1126±0.0008	0.3590	142.2668
pH'$_A$	0.1059±0.0000	0.0949±0.0007	0.1399±0.0000	0.1223±0.0015	2.5530	1362.2580
SteamGen (y_1)	0.0191±0.0000	0.0190±0.0000	0.1446±0.0000	0.1469±0.0020	4.5100	2758.6328
SteamGen (y_2)	0.2303±0.0000	0.2289±0.0002	0.2732±0.0000	0.2732±0.0009	27.3960	10316.5824
SteamGen (y_3)	0.0627±0.0000	0.0626±0.0000	0.1440±0.0000	0.1440±0.0003	0.0640	31.7764
SteamGen (y_4)	0.0467±0.0000	0.0464±0.0000	0.1265±0.0000	0.1243±0.0003	16.0590	10078.0638
Robot (τ_1)	0.0774±0.0000	0.0735±0.0001	0.0840±0.0000	0.0793±0.0003	6.4780	1443.3162
Robot (τ_2)	0.0775±0.0000	0.0699±0.0003	0.1523±0.0000	0.1107±0.0019	71.1330	19965.3736
Robot (τ_3)	0.0751±0.0000	0.0673±0.0004	0.1029±0.0000	0.0914±0.0010	80.3740	17980.0436
Robot (τ_4)	0.0542±0.0000	0.0538±0.0000	0.0593±0.0000	0.0583±0.0002	1.8950	706.0500
Robot (τ_5)	0.1638±0.0000	0.1591±0.0004	0.2421±0.0000	0.2186±0.0013	3.3720	956.9910
Robot (τ_6)	0.0712±0.0000	0.0714±0.0003	0.0962±0.0000	0.0917±0.0004	4.6360	1187.6196
Robot (τ_7)	0.0565±0.0000	0.0563±0.0001	0.0646±0.0000	0.0639±0.0003	2.0560	570.6176
Winding (T_1)	0.2837±0.0000	0.2700±0.0009	0.3359±0.0000	0.3213±0.0006	4.7660	2134.0976
Winding (T_2)	0.0813±0.0000	0.0793±0.0004	0.1893±0.0000	0.1770±0.0016	1.8630	605.5982

Table A.21: Comparing the single-output performance of L++ model trees and Bagging of L++ model trees.

	OSA		**SIM**		**time**	
	L++	**Bagg L++**	**L++**	**Bagg L++**	**L++**	**Bagg L++**
CSTR (Ca)	0.0018±0.0000	0.0018±0.0000	0.0031±0.0000	0.0031±0.0001	15.3010	3835.3238
CSTR' (Ca)	0.0177±0.0000	0.0154±0.0002	0.0195±0.0000	0.0168±0.0002	3.8280	639.3932
CSTR (T)	0.0021±0.0000	0.0021±0.0000	0.0039±0.0000	0.0041±0.0001	12.9890	2047.6336
CSTR' (T)	0.0113±0.0000	0.0100±0.0001	0.0120±0.0000	0.0106±0.0001	2.1960	604.7648
GLS (h_1)	0.0045±0.0000	0.0042±0.0000	0.1139±0.0000	0.1820±0.0055	0.5490	145.8426
GLS (p_1)	0.1036±0.0000	0.1043±0.0002	0.1693±0.0000	0.1709±0.0006	0.7810	220.2894
Narendra	0.0342±0.0000	0.0355±0.0002	0.0680±0.0000	0.0694±0.0006	3.2880	894.8270
Narendra'	0.0693±0.0000	0.0695±0.0001	0.1235±0.0000	0.1238±0.0001	0.7320	317.8562
pH$_B$	0.0678±0.0000	0.0556±0.0007	0.1175±0.0000	0.1389±0.0035	0.6710	370.0352
pH'$_B$	0.0770±0.0000	0.0790±0.0009	0.2108±0.0000	0.2175±0.0032	0.7970	101.4058
pH$_A$	0.0578±0.0000	0.0514±0.0006	0.0990±0.0000	0.0844±0.0008	1.9880	773.0624
pH'$_A$	0.0960±0.0000	0.0898±0.0007	0.1183±0.0000	0.1162±0.0015	1.4440	1021.2264
SteamGen (y_1)	0.0205±0.0000	0.0194±0.0000	0.1829±0.0000	0.1680±0.0021	108.2570	46130.8670
SteamGen (y_2)	0.2320±0.0000	0.2252±0.0004	0.2739±0.0000	0.2642±0.0011	82.3510	23639.0056
SteamGen (y_3)	0.0627±0.0000	0.0626±0.0000	0.1440±0.0000	0.1438±0.0003	16.0140	13890.5426
SteamGen (y_4)	0.0499±0.0000	0.0499±0.0000	0.1540±0.0000	0.1545±0.0002	0.0400	17.8836
Robot (τ_1)	0.0750±0.0000	0.0682±0.0001	0.0797±0.0000	0.0711±0.0002	4.3450	560.9424
Robot (τ_2)	0.0812±0.0000	0.0714±0.0001	0.1733±0.0000	0.1169±0.0014	5.4110	2029.3106
Robot (τ_3)	0.0654±0.0000	0.0627±0.0001	0.1303±0.0000	0.1126±0.0012	1.6870	687.9430
Robot (τ_4)	0.0583±0.0000	0.0536±0.0002	0.0605±0.0000	0.0556±0.0002	6.1480	1631.5214
Robot (τ_5)	0.1594±0.0000	0.1537±0.0003	0.2142±0.0000	0.1986±0.0005	1.3040	430.7552
Robot (τ_6)	0.0730±0.0000	0.0694±0.0003	0.0971±0.0000	0.0870±0.0005	2.4960	856.1418
Robot (τ_7)	0.0571±0.0000	0.0522±0.0001	0.0670±0.0000	0.0571±0.0002	2.6430	369.7446
Winding (T_1)	0.2749±0.0000	0.2659±0.0009	0.3251±0.0000	0.3198±0.0014	5.2140	1783.2740
Winding (T_2)	0.0864±0.0000	0.0688±0.0006	0.1729±0.0000	0.1421±0.0010	16.8360	4826.6712

Table A.22: Comparison of Model Tree Ensembles vs. Neural Networks and ANFIS. Prediction errors and output errors are shown.

	OSA					SIM				
	Bagg.L++	**BMT**	**FMT**	**NN**	**ANFIS**	**Bagg.L++**	**BMT**	**FMT**	**NN**	**ANFIS**
CSTR (Ca)	0.0018±0.0000	0.0055±0.0014	0.0046±0.0005	0.0026±0.0000	0.0017±0.0000	0.0031±0.0000	0.0072±0.0018	0.0064±0.0008	0.0037±0.0000	0.0021±0.0000
CSTR' (Ca)	0.0154±0.0000	0.0164±0.0013	0.0153±0.0009	0.0143±0.0000	0.0258±0.0038	0.0168±0.0000	0.0181±0.0014	0.0170±0.0011	0.0148±0.0000	0.0274±0.0100
CSTR (T)	0.0021±0.0000	0.0064±0.0002	0.0054±0.0003	0.0045±0.0000	0.0040±0.0000	0.0041±0.0000	0.0113±0.0003	0.0089±0.0003	0.0068±0.0000	0.0063±0.0000
CSTR' (T)	0.0100±0.0000	0.0225±0.0012	0.0200±0.0019	0.0191±0.0000	0.0229±0.0001	0.0106±0.0000	0.0241±0.0014	0.0213±0.0019	0.0199±0.0000	0.0241±0.0000
GLS (h_1)	0.0042±0.0000	0.0049±0.0001	0.0043±0.0008	0.0048±0.0000	0.0042±0.0000	0.1820±0.0100	0.1201±0.0263	0.2236±0.0727	0.1312±0.0200	0.5068±0.0000
GLS (p_1)	0.1043±0.0000	0.1201±0.0011	0.1160±0.0113	0.1196±0.0200	0.0974±0.0000	0.1709±0.0000	0.2045±0.0032	0.1809±0.0034	0.2284±0.0700	0.1477±0.0000
Narendra	0.0355±0.0000	0.0270±0.0021	1.9467±2.5247	0.0008±0.0000	0.0097±0.0000	0.0694±0.0000	0.0399±0.0035	0.0440±0.0164	0.0020±0.0000	0.0283±0.0000
Narendra'	0.0695±0.0000	0.0686±0.0035	0.0637±0.0036	0.0388±0.0000	0.0408±0.0000	0.1238±0.0000	0.1256±0.0053	0.1138±0.0070	0.0860±0.0100	0.0787±0.0000
pH$_B$	0.0556±0.0000	0.0932±0.0179	0.0745±0.0065	0.0641±0.0200	0.0704±0.0001	0.1389±0.0000	0.2172±0.0188	0.1951±0.0194	0.1908±0.1000	0.2013±0.0000
pH'$_B$	0.0790±0.0000	0.0838±0.0026	0.0765±0.0011	0.1332±0.0900	0.0821±0.0149	0.2175±0.0000	0.2110±0.0077	0.1973±0.0055	0.2652±0.0700	0.2120±0.0500
pH$_A$	0.0514±0.0000	0.0550±0.0023	0.0550±0.0036	0.0805±0.0200	0.0473±0.0000	0.0844±0.0000	0.1098±0.0058	0.1013±0.0053	0.2118±0.0700	0.0766±0.0000
pH'$_A$	0.0898±0.0000	0.1043±0.0053	0.0974±0.0166	0.1239±0.0200	0.0949±0.0000	0.1162±0.0000	0.1856±0.0157	0.1314±0.0300	0.2250±0.0500	0.1284±0.0000
SteamGen (y_1)	0.0194±0.0000	0.0195±0.0002	0.0193±0.0001	0.0198±0.0000	0.0195±0.0000	0.1680±0.0000	0.1444±0.0074	0.1407±0.0052	0.1749±0.0500	0.1495±0.0000
SteamGen (y_2)	0.2252±0.0000	0.2190±0.0029	0.2216±0.0040	0.2230±0.0000	0.2468±0.0000	0.2642±0.0000	0.2488±0.0083	0.2542±0.0100	0.2421±0.0000	0.3813±0.0000
SteamGen (y_3)	0.0626±0.0000	0.0655±0.0010	0.0674±0.0035	0.0639±0.0000	0.0643±0.0000	0.1438±0.0000	0.1497±0.0017	0.1538±0.0087	0.1516±0.0000	0.1576±0.0000
SteamGen (y_4)	0.0499±0.0000	0.0505±0.0003	0.0505±0.0003	0.0504±0.0000	0.0502±0.0000	0.1545±0.0000	0.1504±0.0004	0.1504±0.0004	0.1514±0.0100	0.1746±0.0000
Robot (τ_1)	0.0682±0.0000	0.0886±0.0009	0.0896±0.0201	0.0977±0.0100	0.0992±0.0000	0.0711±0.0000	0.1675±0.0397	0.0967±0.0069	3.3860±7.3200	0.1208±0.0000
Robot (τ_2)	0.0714±0.0000	0.1036±0.0019	0.1770±0.0113	0.1047±0.0100	0.2152±0.0484	0.1169±0.0000	NaN	0.2121±0.0196	1.2849±1.3800	1.0153±0.2100
Robot (τ_3)	0.0627±0.0000	0.0720±0.0008	0.1591±0.0642	0.0863±0.0100	0.0997±0.0053	0.1126±0.0000	0.2149±0.0033	0.2147±0.0387	0.2856±0.1400	0.1415±0.0100
Robot (τ_4)	0.0536±0.0000	0.0690±0.0014	0.0592±0.0021	0.0745±0.0100	0.0827±0.0081	0.0556±0.0000	0.1904±0.0095	0.0772±0.0068	0.8769±1.6700	0.0763±0.0100
Robot (τ_5)	0.1537±0.0000	0.1897±0.0001	0.1704±0.0096	0.2385±0.0600	0.1859±0.0071	0.1986±0.0000	0.3660±0.0044	0.2513±0.0267	0.3208±0.1400	0.2395±0.0100
Robot (τ_6)	0.0694±0.0000	0.0886±0.0039	0.1178±0.0871	0.0829±0.0000	0.0802±0.0013	0.0870±0.0000	0.1613±0.0135	0.1225±0.0201	0.1674±0.0300	0.1384±0.0000
Robot (τ_7)	0.0522±0.0000	0.0603±0.0002	0.0558±0.0017	0.0707±0.0000	0.0624±0.0000	0.0571±0.0000	0.0709±0.0016	0.0649±0.0028	0.0867±0.0100	0.0972±0.0000
Winding (T_1)	0.2659±0.0000	0.2818±0.0080	0.2753±0.0108	0.3022±0.0300	0.3150±0.0366	0.3198±0.0000	0.3477±0.0050	0.3445±0.0150	0.5407±0.3000	0.5817±0.3800
Winding (T_2)	0.0688±0.0000	0.1095±0.0088	0.0863±0.0043	0.0940±0.0100	0.0788±0.0001	0.1421±0.0000	0.2897±0.0596	0.1839±0.0334	0.2064±0.0400	0.1677±0.0000

Table A.23: Comparison of Model Tree Ensembles vs. Neural Networks and ANFIS. Model complexities and times required for learning are shown.

	complexity					**time**				
	Bagg.L++	**BMT**	**FMT**	**NN**	**ANFIS**	**Bagg.L++**	**BMT**	**FMT**	**NN**	**ANFIS**
CSTR (Ca)	30.00	9.95	28.16	15.00	2.00	3835.32	80.03	55.68	3.90	4790.15
CSTR' (Ca)	14.00	2.90	2.65	4.00	6.00	639.39	116.83	83.78	2.60	578.73
CSTR (T)	27.00	1.19	3.64	3.00	3.00	2047.63	53.76	40.08	13.43	1791.55
CSTR' (T)	10.00	2.00	1.56	6.00	4.00	604.76	120.10	64.36	0.92	205.41
GLS (h_1)	12.00	1.62	1.90	1.00	2.00	145.84	5.90	3.25	2.96	249.29
GLS (p_1)	16.00	4.67	5.28	3.00	2.00	220.29	7.78	8.52	1.53	41.17
Narendra	30.00	67.67	76.42	15.00	3.00	894.83	13.30	9.52	16.37	1092.73
Narendra'	12.00	4.00	5.02	7.00	3.00	317.86	36.83	36.80	0.55	1138.78
pH$_B$	18.00	5.86	5.11	8.00	4.00	370.04	4.57	5.02	0.49	5.70
pH'$_B$	12.00	4.90	1.41	4.00	5.00	101.41	9.23	7.98	0.24	7.93
pH$_A$	29.00	6.24	5.63	2.00	3.00	773.06	4.42	3.32	0.34	29.89
pH'$_A$	29.00	1.00	1.19	4.00	3.00	1021.23	5.60	3.67	0.18	201.76
SteamGen (y_1)	28.00	1.00	1.00	2.00	2.00	46130.87	99.70	101.97	3.32	715.67
SteamGen (y_2)	30.00	16.71	37.18	1.00	2.00	23639.01	198.22	227.33	0.47	702.18
SteamGen (y_3)	10.00	4.33	32.85	1.00	2.00	13890.54	798.72	733.95	5.36	5781.82
SteamGen (y_4)	1.00	1.05	6.04	1.00	3.00	17.88	162.63	174.57	1.67	1333.78
Robot (τ_1)	11.00	1.00	4.38	4.00	3.00	560.94	30.83	23.27	0.52	219.42
Robot (τ_2)	13.00	85.90	15.01	7.00	10.00	2029.31	48.88	38.17	0.47	1640.93
Robot (τ_3)	6.00	1.00	8.50	13.00	6.00	687.94	47.20	40.33	0.97	633.16
Robot (τ_4)	18.00	1.00	1.69	10.00	6.00	1631.52	36.73	28.03	0.57	657.80
Robot (τ_5)	5.00	6.38	1.53	13.00	5.00	430.75	31.65	17.23	0.75	3934.20
Robot (τ_6)	8.00	1.00	1.45	2.00	2.00	856.14	152.17	69.75	0.63	81.43
Robot (τ_7)	8.00	1.00	1.28	11.00	2.00	369.75	131.27	77.92	0.85	3943.97
Winding (T_1)	14.00	2.90	4.40	5.00	3.00	1783.27	144.45	155.68	0.72	144.14
Winding (T_2)	28.00	2.24	2.90	10.00	2.00	4826.67	265.47	249.07	1.27	3733.28

Table A.24: Comparing the performance of multiple-output L++ and Bagging of multiple-output L++ model trees. The table reports identical numbers under the time column, for each of the outputs of a dynamic system model.

	OSA		**SIM**		**time**	
	L++$_{MO}$	**Bagg L++$_{MO}$**	**L++$_{MO}$**	**Bagg L++$_{MO}$**	**L++$_{MO}$**	**Bagg L++$_{MO}$**
CSTR (Ca)	0.0019±0.0000	0.0020±0.0000	0.0166±0.0000	0.0159±0.0002	18.6060	5089.3000
CSTR (T)	0.0022±0.0000	0.0022±0.0000	0.0134±0.0000	0.0125±0.0001	18.6060	5089.3000
CSTR' (Ca)	0.0133±0.0000	0.0110±0.0001	0.0191±0.0000	0.0207±0.0005	7.3520	2111.0930
CSTR' (T)	0.0136±0.0000	0.0112±0.0001	0.0200±0.0000	0.0181±0.0002	7.3520	2111.0930
GLS (p_1)	0.0923±0.0000	0.0889±0.0005	0.2370±0.0000	0.2326±0.0010	4.1950	2427.0580
GLS (h_1)	0.0040±0.0000	0.0038±0.0000	0.1462±0.0000	0.1414±0.0025	4.1950	2427.0580
SteamGen (y_1)	0.0194±0.0000	0.0191±0.0000	0.1573±0.0000	0.1574±0.0009	117.2630	30347.2690
SteamGen (y_2)	0.2373±0.0000	0.2332±0.0002	0.2878±0.0000	0.2897±0.0012	117.2630	30347.2690
SteamGen (y_3)	0.0643±0.0000	0.0636±0.0000	0.1999±0.0000	0.2002±0.0005	117.2630	30347.2690
SteamGen (y_4)	0.0468±0.0000	0.0467±0.0001	0.1365±0.0000	0.1380±0.0006	117.2630	30347.2690
Robot (τ_1)	0.0879±0.0000	0.0691±0.0003	0.0995±0.0000	0.0745±0.0006	19.8540	4005.1430
Robot (τ_2)	0.0918±0.0000	0.0724±0.0003	0.1745±0.0000	0.1494±0.0022	19.8540	4005.1430
Robot (τ_3)	0.0768±0.0000	0.0603±0.0001	0.1102±0.0000	0.0981±0.0009	19.8540	4005.1430
Robot (τ_4)	0.0618±0.0000	0.0495±0.0002	0.0812±0.0000	0.0535±0.0004	19.8540	4005.1430
Robot (τ_5)	0.1928±0.0000	0.1437±0.0006	0.2301±0.0000	0.1778±0.0008	19.8540	4005.1430
Robot (τ_6)	0.0734±0.0000	0.0662±0.0006	0.0996±0.0000	0.0904±0.0006	19.8540	4005.1430
Robot (τ_7)	0.0575±0.0000	0.0497±0.0001	0.0777±0.0000	0.0573±0.0005	19.8540	4005.1430
Winding (T_1)	0.2823±0.0000	0.2681±0.0014	0.3687±0.0000	0.3500±0.0015	10.9080	1449.9782
Winding (T_2)	0.1045±0.0000	0.0848±0.0005	0.2149±0.0000	0.1791±0.0006	10.9080	1449.9782

Table A.25: Comparing single-output model trees, each predicting one output variable, to one multi-output model tree.

	OSA		**SIM**		**num.LMs**		**time**	
	Sep.SO L++	**L++$_{MO}$**	**Sep.SO L++**	**L++$_{MO}$**	**Sep.SO L++**	**L++$_{MO}$**	**Sep.SO L++**	**L++$_{MO}$**
CSTR (Ca)	0.0018	0.0019	0.0130	0.0166	57.0000	30.0000	25.8690	18.6060
CSTR (T)	0.0024	0.0022	0.0114	0.0134	57.0000	30.0000	25.8690	18.6060
CSTR' (Ca)	0.0171	0.0133	NaN	0.0191	24.0000	17.0000	4.9050	7.3520
CSTR' (T)	0.0142	0.0136	NaN	0.0200	24.0000	17.0000	4.9050	7.3520
GLS (p_1)	0.2275	0.0923	1.2334	0.2370	39.0000	30.0000	3.5660	4.1950
GLS (h_1)	2.3451	0.0040	2.7523	0.1462	39.0000	30.0000	3.5660	4.1950
SteamGen (y_1)	0.0200	0.0194	0.2111	0.1573	69.0000	21.0000	189.5580	117.2630
SteamGen (y_2)	0.2426	0.2373	0.2982	0.2878	69.0000	21.0000	189.5580	117.2630
SteamGen (y_3)	0.0646	0.0643	0.2222	0.1999	69.0000	21.0000	189.5580	117.2630
SteamGen (y_4)	0.0499	0.0468	0.1632	0.1365	69.0000	21.0000	189.5580	117.2630
Robot (τ_1)	0.0751	0.0879	0.1143	0.0995	69.0000	14.0000	19.9880	19.8540
Robot (τ_2)	0.0831	0.0918	0.3348	0.1745	69.0000	14.0000	19.9880	19.8540
Robot (τ_3)	0.0655	0.0768	0.1747	0.1102	69.0000	14.0000	19.9880	19.8540
Robot (τ_4)	0.0677	0.0618	0.0721	0.0812	69.0000	14.0000	19.9880	19.8540
Robot (τ_5)	0.1622	0.1928	0.3019	0.2301	69.0000	14.0000	19.9880	19.8540
Robot (τ_6)	0.0708	0.0734	0.1439	0.0996	69.0000	14.0000	19.9880	19.8540
Robot (τ_7)	0.0571	0.0575	0.0729	0.0777	69.0000	14.0000	19.9880	19.8540
Winding (T_1)	0.2770	0.2823	0.3565	0.3687	42.0000	19.0000	16.9240	10.9080
Winding (T_2)	0.0780	0.1045	0.1685	0.2149	42.0000	19.0000	16.9240	10.9080

Table A.26: Comparing a separate ensemble of single-output L++ MTs, and a single ensemble of multiple-output L++ MTs.

	OSA		**SIM**		**num.LM**		**time**	
	Sep.Bagg.L++	**Bagg.L++$_{MO}$**	**Sep.Bagg.L++**	**Bagg.L++$_{MO}$**	**Sep.Bagg.L++**	**Bagg.L++$_{MO}$**	**Sep.Bagg.L++**	**Bagg.L++$_{MO}$**
CSTR (Ca)	0.0018±0.0000	0.0020±0.0000	0.0129±0.0001	0.0159±0.0002	57.0000	30.0000	20285.2790	5089.3000
CSTR (T)	0.0022±0.0000	0.0022±0.0000	0.0109±0.0000	0.0125±0.0001	57.0000	30.0000	20285.2790	5089.3000
CSTR' (Ca)	0.0135±0.0003	0.0110±0.0001	NaN	0.0207±0.0005	24.0000	17.0000	1593.0990	2111.0930
CSTR' (T)	0.0128±0.0003	0.0112±0.0001	NaN	0.0181±0.0002	24.0000	17.0000	1593.0990	2111.0930
GLS (p_1)	0.2140±0.0007	0.0889±0.0005	1.3191±0.0458	0.2326±0.0010	39.0000	30.0000	1172.0410	2427.0580
GLS (h_1)	1.8919±0.1130	0.0038±0.0000	3.3355±0.0823	0.1414±0.0025	39.0000	30.0000	1172.0410	2427.0580
SteamGen (y_1)	0.0193±0.0000	0.0191±0.0000	0.1881±0.0020	0.1574±0.0009	69.0000	21.0000	52196.4740	30347.2690
SteamGen (y_2)	0.2276±0.0004	0.2332±0.0002	0.2751±0.0013	0.2897±0.0012	69.0000	21.0000	52196.4740	30347.2690
SteamGen (y_3)	0.0623±0.0000	0.0636±0.0000	0.2149±0.0002	0.2002±0.0005	69.0000	21.0000	52196.4740	30347.2690
SteamGen (y_4)	0.0499±0.0000	0.0467±0.0001	0.1616±0.0004	0.1380±0.0006	69.0000	21.0000	52196.4740	30347.2690
Robot (τ_1)	0.0699±0.0002	0.0691±0.0003	0.0836±0.0003	0.0745±0.0006	69.0000	14.0000	12153.7380	4005.1430
Robot (τ_2)	0.0757±0.0003	0.0724±0.0003	0.1877±0.0023	0.1494±0.0022	69.0000	14.0000	12153.7380	4005.1430
Robot (τ_3)	0.0632±0.0001	0.0603±0.0001	0.1205±0.0008	0.0981±0.0009	69.0000	14.0000	12153.7380	4005.1430
Robot (τ_4)	0.0546±0.0007	0.0495±0.0002	0.0602±0.0006	0.0535±0.0004	69.0000	14.0000	12153.7380	4005.1430
Robot (τ_5)	0.1587±0.0001	0.1437±0.0006	0.2255±0.0015	0.1778±0.0008	69.0000	14.0000	12153.7380	4005.1430
Robot (τ_6)	0.0692±0.0004	0.0662±0.0006	0.1012±0.0011	0.0904±0.0006	69.0000	14.0000	12153.7380	4005.1430
Robot (τ_7)	0.0521±0.0001	0.0497±0.0001	0.0620±0.0004	0.0573±0.0005	69.0000	14.0000	12153.7380	4005.1430
Winding (T_1)	0.2618±0.0009	0.2681±0.0014	0.3426±0.0023	0.3500±0.0015	42.0000	19.0000	4838.2250	1449.9782
Winding (T_2)	0.0687±0.0008	0.0848±0.0005	0.1551±0.0006	0.1791±0.0006	42.0000	19.0000	4838.2250	1449.9782

References

Aizerman, A., Braverman, E. M., & Rozoner, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, *25*, 821–837.

Aleksovski, D., Kocev, D., & Džeroski, S. (2009). Evaluation of distance measures for hierarchical multilabel classification in functional genomics. In *1st Workshop on Learning from Multi-Label Data (MLD)* (pp. 5–16).

Aleksovski, D., Kocijan, J., & Džeroski, S. (2013). Model tree ensembles for modeling dynamic systems. *Lecture notes in computer science*, *8140*, 17–32.

Aleksovski, D., Kocijan, J., & Džeroski, S. (2014a). *Ensembles of linear model trees for the identification of multiple-output systems.*

Aleksovski, D., Kocijan, J., & Džeroski, S. (2014b). Model tree ensembles for the identification of multiple-output systems. In *2014 European Control Conference (ECC)* (pp. 750–755). IEEE.

Aleksovski, D., Kocijan, J., & Džeroski, S. (2014c). Model-Tree Ensembles for noise-tolerant system identification. *Advanced Engineering Informatics*. doi:http://dx.doi.org/10.1016/j.aei.2014.07.008

Alexander, W. P. & Grimshaw, S. D. (1996). Treed regression. *Journal of Computational and Graphical Statistics*, *5*(2), 156–175.

Appice, A. & Džeroski, S. (2007). Stepwise induction of multi-target model trees. In *Proceedings of the 18th European Conference on Machine Learning* (pp. 502–509). Springer.

Asuncion, A. & Newman, D. (2007). Uci machine learning repository. Retrieved from http://archive.ics.uci.edu/ml%20(Accessed:%20June%202013)

Ažman, K. & Kocijan, J. (2011). Dynamical systems identification using Gaussian process models with incorporated local models. *Engineering Applications of Artificial Intelligence*, *24*(2), 398–408.

Bastogne, T., Garnier, H., & Sibille, P. (2001). A pmf-based subspace method for continuous-time model identification. application to a multivariable winding process. *International Journal of Control*, *74*(2), 118–132.

Billings, S. (2013). *Nonlinear system identification: narmax methods in the time, frequency, and spatio-temporal domains*. John Wiley & Sons.

Billings, S. & Voon, W. (1987). Piecewise linear identification of non-linear systems. *International journal of control*, *46*(1), 215–235.

Bishop, C. M. et al. (2006). *Pattern recognition and machine learning*. springer New York.

Blockeel, H. & De Raedt, L. (1998). Top-down induction of first-order logical decision trees. *Artificial intelligence*, *101*(1), 285–297.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational Learning Theory* (pp. 144–152). ACM.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Wadsworth Inc.

Connally, P., Li, K., & Irwin, G. W. (2007). Prediction- and simulation-error based perceptron training: solution space analysis and a novel combined training scheme. *Neurocomputing*, *70*(4–6), 819–827. Advanced Neurocomputing Theory and Methodology Selected papers from the International Conference on Intelligent Computing 2005 (ICIC 2005) International Conference on Intelligent Computing 2005. doi:http://dx.doi.org/10.1016/j.neucom.2006.10.013

Cristianini, N. & Shawe-Taylor, J. (2000). *An introduction to support vector machines and other kernel-based learning methods*. Cambridge University Press.

De Moor, B. (2013). Database for the identification of systems (DaISy). Department of Electrical Engineering, ESAT/SCD, KU Leuven, Belgium, http://www.esat.kuleuven.ac.be/sista/daisy. Accessed September 2013.

Demšar, D., Debeljak, M., Lavigne, C., & Džeroski, S. (2005). Modelling pollen dispersal of genetically modified oilseed rape within the field, 2005. In *Abstract papers at The Annual Meeting of the Ecological Society of America* (pp. 7–12).

Dobra, A. & Gehrke, J. (2002). Secret: a scalable linear regression tree algorithm. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 481–487). ACM.

Džeroski, S., Demšar, D., & Grbović, J. (2000). Predicting chemical parameters of river water quality from bioindicator data. *Applied Intelligence*, *13*(1), 7–17.

Espinosa, J. J. & Vandewalle, J. (1999). Predictive control using fuzzy models. In *Advances in soft computing* (pp. 187–200). Springer.

Frank, E., Wang, Y., Inglis, S., Holmes, G., & Witten, I. H. (1998). Using model trees for classification. *Machine Learning*, *32*(1), 63–76.

Gama, J. (2004). Functional trees. *Machine Learning*, *55*(3), 219–250.

Giri, F. & Bai, E.-W. (2010). *Block-oriented nonlinear system identification*. Lecture Notes in Control and Information Sciences. Springer.

Grandvalet, Y. (2004). Bagging equalizes influence. *Machine Learning*, *55*(3), 251–270.

Haber, R. & Unbehauen, H. (1990). Structure identification of nonlinear dynamic systems-a survey on input/output approaches. *Automatica*, *26*(4), 651–677.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations*, *11*(1), 10–18.

Henson, M. A. & Seborg, D. E. (1994). Adaptive nonlinear control of a pH neutralization process. *IEEE Transactions on Control Systems Technology*, *2*(3), 169–182.

Hoffmann, F. & Nelles, O. (2001). Genetic programming for model selection of tsk-fuzzy systems. *Information Sciences*, *136*(1–4), 7–28. doi:http://dx.doi.org/10.1016/S0020-0255(01)00139-6

Ikonomovska, E. (2012). *Algorithms for learning regression trees and ensembles on evolving data streams* (Doctoral dissertation, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia).

Ikonomovska, E., Gama, J., & Džeroski, S. (2011). Learning model trees from evolving data streams. *Data mining and knowledge discovery*, *23*(1), 128–168.

Jang, J.-S. R. (1994). Structure determination in fuzzy modeling: a fuzzy CART approach. In *Proceedings of the Third IEEE Conference on Fuzzy Systems* (pp. 480–485). IEEE.

Jang, J.-S. R., Sun, C.-T., & Mizutani, E. (1997). *Neuro-fuzzy and soft computing–a computational approach to learning and machine intelligence*. Prentice Hall.

Johansen, T. & Babuška, R. (2003). Multiobjective identification of Takagi-Sugeno fuzzy models. *IEEE Transactions on Fuzzy Systems*, *11*(6), 847–860.

Johansen, T. & Foss, B. A. (1995). Identification of non-linear system structure and parameters using regime decomposition. *Automatica*, *31*(2), 321–326.

Johansen, T. & Foss, B. A. (1997). Operating regime based process modeling and identification. *Computers & Chemical Engineering*, *21*(2), 159–176.

Jung, M., Reichstein, M., & Bondeau, A. (2009). Towards global empirical upscaling of fluxnet eddy covariance observations: validation of a model tree ensemble approach using a biosphere model. *Biogeosciences*, *6*(10), 2001–2013.

Kampichler, C., Džeroski, S., & Wieland, R. (2000). Application of machine learning techniques to the analysis of soil ecological data bases: relationships between habitat features and collembolan community characteristics. *Soil Biology and Biochemistry*, *32*(2), 197–209.

Karalič, A. (1992). Employing linear regression in regression tree leaves. In *Proceedings of the 10th European Conference on Artificial Intelligence* (pp. 440–441). John Wiley & Sons.

Karalič, A. & Bratko, I. (1997). First order regression. *Machine Learning*, *26*(2-3), 147–176.

Keesman, K. J. (2011). *System identification: an introduction*. Springer.

Kocev, D., Struyf, J., & Džeroski, S. (2006). Beam search induction and similarity constraints for predictive clustering trees. In *Proceedings of the 5th International Conference on Knowledge Discovery in Inductive Databases* (pp. 134–151). Springer.

Kocijan, J. & Grancharova, A. (2010). Gaussian process modelling case study with multiple outputs. *Comptes Rendus de l'Academie Bulgare des Sciences*, *63*, 601–607.

Kocijan, J. & Likar, B. (2008). Gas–liquid separator modelling and simulation with Gaussian-process models. *Simulation Modelling Practice and Theory*, *16*(8), 910–922.

Kocijan, J. & Petelin, D. (2011). Output-error model training for Gaussian process models. In A. Dobnikar, U. Lotrič, & B. Šter (Eds.), *Adaptive and natural computing algorithms* (Vol. 6594, pp. 312–321). Lecture Notes in Computer Science. Springer.

Krogh, A. & Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In *Proceedings of the 8th International Conference on Advances in Neural Information Processing Systems* (pp. 231–238). MIT Press.

Lemos, A., Caminhas, W., & Gomide, F. (2011). Evolving fuzzy linear regression trees with feature selection. In *Proceedings of the IEEE Workshop on Evolving and Adaptive Intelligent Systems* (pp. 31–38). IEEE.

Lightbody, G. & Irwin, G. W. (1997). Nonlinear control structures based on embedded neural system models. *IEEE Transactions on Neural Networks*, *8*(3), 553–567.

Ljung, L. (1987). *System identification: theory for the user*. Prentice Hall.

Loh, W.-Y. (2002). Regression trees with unbiased variable selection and interaction detection. *Statistica Sinica*, *12*(2), 361–386.

Malerba, D., Esposito, F., Ceci, M., & Appice, A. (2004). Top-down induction of model trees with regression and splitting nodes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *26*(5), 612–625.

Marsala, C. (2009). Data mining with ensembles of fuzzy decision trees. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining* (pp. 348–354). IEEE.

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133.

Murray-Smith, R. & Johansen, T. (Eds.). (1997). *Multiple model approaches to modelling and control*. Taylor & Francis.

Narendra, K. S. & Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, *1*(1), 4–27.

Nelles, O. (1995). On training radial basis function networks as series-parallel and parallel models for identification of nonlinear dynamic systems. In *Proceedings of the 1995 IEEE International Conference on Systems, Man and Cybernetics* (pp. 4609–4614). IEEE.

Nelles, O. (1996). Local linear model trees for on-line identification of time-variant nonlinear dynamic systems. In *Proceedings of the 1996 artificial neural networks conference (ICANN 96)* (pp. 115–120). Springer.

Nelles, O. (1999). *Nonlinear system identification with local linear neuro-fuzzy models.* Shaker.

Nelles, O. (2001). *Nonlinear system identification: from classical approaches to neural networks and fuzzy models.* Springer.

Nelles, O. (2006). Axes-oblique partitioning strategies for local model networks. In *Proceedings of the 2006 ieee international symposium on intelligent control* (pp. 2378–2383). IEEE.

Olaru, C. & Wehenkel, L. (2003). A complete fuzzy decision tree technique. *Fuzzy Sets and Systems*, *138*(2), 221–254.

Nonlinar System Identification. (2014). Retrieved August 1, 2014, from http://en.wikipedia.org/wiki/Nonlinear_system_identification

Pellegrinetti, G. & Bentsman, J. (1996). Nonlinear control oriented boiler modeling-a benchmark problem for controller design. *IEEE Transactions on Control Systems Technology*, *4*(1), 57–64.

Pfahringer, B. (2011). Semi-random model tree ensembles: an effective and scalable regression method. In *Proceeding of the 2011 Australasian Conference on Artificial Intelligence* (pp. 231–240).

Piroddi, L. [L.] & Spinelli, W. (2003). An identification algorithm for polynomial narx models based on simulation error minimization. *International Journal of Control*, *76*(17), 1767–1781.

Piroddi, L. [Luigi]. (2008). Simulation error minimisation methods for narx model identification. *International Journal of Modelling, Identification and Control*, *3*(4), 392–403.

Potts, D. & Sammut, C. (2007). *Learning to control* (Doctoral dissertation, School of Computer Science and Engineering, University of New South Wales).

Quinlan, J. R. (1992). Learning with continuous classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence* (Vol. 92, pp. 343–348). World Scientific.

Rasmussen, C. E. & Williams, C. K. I. (2006). *Gaussian processes for machine learning.* MIT Press.

Rumelhart, D. E., Hintont, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.

Segal, M. R. (2004). *Machine learning benchmarks and random forest regression.* Center for Bioinformatics and Molecular Biostatistics, University of California.

Stojanova, D. (2009). *Estimating forest properties from remotely sensed data by using machine learning* (Master's thesis, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia).

Suarez, A. & Lutsko, J. (1999). Globally optimal fuzzy decision trees for classification and regression. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *21*(12), 1297–1311.

Takagi, T. & Sugeno, M. (1985). Fuzzy identification of systems and its applications to modeling and control. *IEEE Transactions on Systems, Man and Cybernetics*, (1), 116–132.

Torgo, L. (1997). Functional models for regression tree leaves. In *Proceedings of 14th the International Conference on Machine Learning* (pp. 385–393). Morgan Kaufmann.

Torgo, L. (2013). Regression datasets. http://www.dcc.fc.up.pt/ ltorgo/Regression/DataSets.html Accessed: June 2013).

Vens, C. & Blockeel, H. (2006). A simple regression based heuristic for learning model trees. *Intelligent Data Analysis*, *10*(3), 215–236.

Vijayakumar, S. & Schaal, S. (2000). Locally weighted projection regression: incremental real time learning in high dimensional space. In *Proceedings of the Seventeenth International Conference on Machine Learning* (pp. 1079–1086). San Francisco, CA, USA: Morgan Kaufmann.

Wang, Y. & Witten, I. H. (1997). Inducing model trees for continuous classes. In *Poster Papers of the 9th European Conference on Machine Learning* (pp. 128–137). Springer.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics bulletin*, 80–83.

Witten, I. H. & Frank, E. (2005). *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann.

Bibliography

Publications Related to the Thesis

Original Scientific Articles

Aleksovski, D., Kocijan, J., & Džeroski, S. (2013). Model tree ensembles for modeling dynamic systems. *Lecture notes in computer science*, *8140*, 17–32.

Aleksovski, D., Kocijan, J., & Džeroski, S. (2014c). Model-Tree Ensembles for noise-tolerant system identification. *Advanced Engineering Informatics*. doi:http://dx.doi.org/10.1016/j.aei.2014.07.008

Published Scientific Conference Contributions

Aleksovski, D., Kocijan, J., & Džeroski, S. (2014b). Model tree ensembles for the identification of multiple-output systems. In *2014 European Control Conference (ECC)* (pp. 750–755). IEEE.

Articles Pending for Publication Related to the Thesis

Aleksovski, D., Kocijan, J., & Džeroski, S. (2014a). *Ensembles of linear model trees for the identification of multiple-output systems.*

Publications not related to the Thesis

Published Scientific Conference Contributions

Aleksovski, D., Kocev, D., & Džeroski, S. (2009). Evaluation of distance measures for hierarchical multilabel classification in functional genomics. In *1st Workshop on Learning from Multi-Label Data (MLD)* (pp. 5–16).

Biography

Darko Aleksovski was born on 10 May 1983 in Skopje, Macedonia. In 2007 he completed the Bachelor of Science degree at the Faculty of Natural Sciences and Mathematics, Ss. Cyril and Methodius University in Skopje. He completed his studies with a grade point average of 9.86 on a 10 scale.

During his primary school, high school and studies he successfully participated in national competitions in mathematics and physics, and national and international competitions in informatics. He won two gold medals on the Macedonian national olympiad in informatics. During high school he held a state scholarship, while during studies he was partly funded by the Institute of Informatics, Faculty of Natural Sciences and Mathematics.

In the fall of 2008 he enrolled in the PhD program "New Media and e-Science" at the Jožef Stefan International Postgraduate School, in Ljubljana, Slovenia. He held a scholarship from the Slovene Human Resources Development and Scholarship Fund. His PhD studies were also partly funded by the Jožef Stefan Institute, through the EU projects PHAGOSYS (Systems biology of phagosome formation and maturation - modulation by intracellular pathogens), SUMO (Supermodeling by combining imperfect models), and MAESTRA (Learning from Massive, Incompletely annotated, and Structured Data).