Ensembles of Fuzzy Linear Model Trees for the Identification of Multi-Output Systems

Darko Aleksovski*, Juš Kocijan†‡§, Sašo Džeroski*‡

*Dept. of Knowledge Technologies, Jožef Stefan Institute, Jamova cesta 39, Ljubljana, Slovenia †Dept. of Systems and Control, Jožef Stefan Institute, Jamova cesta 39, Ljubljana, Slovenia ‡Jožef Stefan International Postgraduate School, Jamova cesta 39, Ljubljana, Slovenia §University of Nova Gorica, Vipavska 13, Nova Gorica, Slovenia

Abstract

We address the task of discrete-time modeling of nonlinear dynamic systems with multiple outputs using measured data. In the area of control engineering, this task is typically converted into a a set of classical regression problem, one for each output, which can then be solved with any nonlinear regression approach. Fuzzy models, in the Takagi-Sugeno form, are popular in this context. We use Lolimot, which is a tree learning method, to build fuzzy linear model trees.

In this paper, we propose, implement and empirically evaluate three extensions of fuzzy linear model trees. First, we consider and evaluate multi-output models. Second, we propose to use ensembles of such models. Third, we investigate the use of a search heuristic based on simulation error (as opposed to the one-step-ahead prediction error), specific to the context of modeling dynamic systems.

We perform an empirical evaluation and compare these approaches on six multi-output case studies, using both measured and simulated data, with noise. The case studies include modeling of the inverse dynamics of a robot arm, as well as five additional process-industry systems.

Ensembles improve the performance of both single and multi-output trees, while the heuristic specific to modeling dynamic systems only improves performance very slightly. Multi-output model trees exhibit comparable or worse predictive performance to a set of single-output models, while providing a more compact model. Overall, we can recommend the use of bagging of single-output Lolimot models, learned by using the simulation error as a search heuristic.

Index Terms

decision tree ensemble, nonlinear dynamic system identification, multiple outputs

I. Introduction

Dynamic systems are systems that 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.

In control engineering, a dynamic system is typically represented by two (sets of) variables. System (endogenous) variables, denoted by y, describe the state that the system is in at any particular point in time. Input (exogenous) variables, denoted by u, capture external conditions that influence, i.e., are relevant to, the system.

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. The task of constructing models of dynamic systems, both in continuous and discrete time, from measured data is the topic of study of the system identification sub-area of control engineering [1], [2].

This paper addresses the task of discrete-time modeling of nonlinear dynamic systems. As mentioned above, two types of variables are used in modeling, system (endogenous) and input (exogenous) variables, denoted by y and u, respectively. Using the *external dynamics approach* [3], 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 state variable y at a given time point k as a function of past system and input variables (y and u). The transformation creates a new vector of features which is composed from the lagged values of the input variable u and system variable y. Typically, up to n time points in the immediate past (with respect to k) are considered. At time point k, the dynamic system is thus represented by the vector of features $\mathbf{x}(k) = [u(k-1), u(k-2), .., u(k-n), y(k-1), y(k-2), .., y(k-n)]^T$ where n is the dynamic order (lag) of the system. The model of the system is a difference equation that describes the state of the system at (a discrete) time point k, $y(k)$ as a function of the previous system states and inputs (i.e., $x(k)$). 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 [4], [5].

One possible approach to the nonlinear function approximation problem is the *multimodel approach* [6]. Its idea is to combine several simple submodels and use them to describe the global behavior of the dynamic system. The operating region of the dynamic system is split into several subregions (partitions) and a simple submodel is learned for each subregion. In the case of multiple outputs, i.e. $y_1, y_2, ..y_r$, the identification problem becomes a challenging task.

This paper considers an automatic procedure for the identification of multiple-output dynamic systems, that uses the multimodel approach. In particular it is concerned with building Takagi-Sugeno (TS) [7] models, using Lolimot fuzzy linear model trees [8], for multiple-output systems. Rather than learning a single model, it learns an ensemble of fuzzy linear model trees.

A baseline approach to the construction of a TS model for a multiple-input multiple-output (MIMO) system is to construct a different multiple-input single-output (MISO) model for each output. In this case, each MISO model is a TS model with different antecedent and consequent parts. In this paper, we compare the baseline approach to the construction of a so called *multi-output TS model* for a MIMO system. The multi-output TS model contains different consequent parts for each of the outputs, and a single common antecedent part.

We investigate the potential benefit of using multi-output (MO) Takagi-Sugeno models. On one hand, the shared antecedents of the fuzzy rules may result in smaller overall complexity of the MIMO model, as compared to the sum of the complexities of the separate MISO models constructed for each output. On the other hand we expect that a similar predictive performance will be achieved in both cases.

The original contribution of this paper is an approach for modeling multi-output dynamic systems with ensembles of Lolimot and its experimental evaluation on six case studies. Lolimot has been extensively used for the identification of SO systems in the literature (e.g. Lolimot, [8]), but its use for the identification of MO systems has been only proposed and not thoroughly investigated. On the other hand, ensembles of such trees have not been considered in the literature.

The remainder of the paper is organized as follows. Section 2 gives a quick overview of related work. The third section introduces the methodology that we propose. Section 4 presents the experimental setup and describes the case studies, while Section 5 presents the results of the experimental evaluation of our approach. Finally, Section 6 concludes and outlines some directions for further work.

II. Related work

Four lines of related work are relevant to the present study. First, we discuss the tree learning procedure in a recursive and iterative manner. Second, we review related work on multi-output trees. Third, we contrast the learning of crisp trees and fuzzy trees. Finally, we discuss the construction of tree ensembles.

A. *Recursive vs iterative construction of model trees*

The machine learning literature provides many different algorithms for learning classification and regression trees [9], and the special case of model trees (the M5' algorithm [10], [11], the HTL algorithm [12], and others [13]–[15]). Axis-orthogonal model trees are defined recursively as follows: 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 (linear) model is defined. An example of a model tree is shown in the left-hand side of Figure 1.

Given the recursive definition of a model tree, most tree learning algorithms operate in a recursive fashion [9], [15]. These algorithms utilize the efficient divide-and-conquer principle: The split node to be added is determined only by using the part of the training data that is in the corresponding partition of the input space. The model trees built in this fashion focus on minimizing prediction error [8], which can be estimated independently for each partition of the tree.

Other algorithms build the tree structure using a different approach. The split node to be added at each step of the method is determined by using the overall (or global) model performance [8], which may include constraints posed on the model [16]. This requires an iterative approach, where in each iteration one more split node is added to the tree structure, i.e., the number of local models is increased by one. At each step, the global performance of the tree as a whole is evaluated. Models built in this way can minimize simulation or output error when modeling dynamic systems [3]. The simulation procedure for evaluating a model of a dynamic system is performed by an iterative application of the predictive model. The predicted value of the system variable y at time k ($\hat{y}(k)$ is fed back as input to the model, instead of a measured value ($y(k)$) at time $k+1$. The resulting model output is called simulation output and the corresponding error is denoted as simulation or output error.

B. Multi-output trees

Motivated by the possibility to use correlations between outputs, several authors [8], [17]–[19] introduce multi-output trees, also called multi-target or multi-objective trees. The terminal nodes in the trees contain local models for several output or target variables. The multi-output modeling possibility, realized as a multi-output linear model tree, is also included in the Lolimot software [3], [20].

The same multi-output idea was also introduced in the context of the ANFIS method for dynamic system identification. The author proposed a modification of the ANFIS method [19] that builds a Takagi-Sugeno model containing different consequent parts for each of the outputs, and a single common antecedent part. We will consider this kind of models in the present paper.

C. Crisp vs fuzzy trees

The commonly used model tree algorithms [14], [21] utilize the recursive divide-and-conquer approach and produce only hard or crisp splits. Another research direction considers trees with soft or fuzzy splits [22], [23]. 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 literature provides many different approaches for learning fuzzy regression trees [24], [25], i.e., trees with constant local models, and a few approaches for learning fuzzy linear model trees [23]. The fuzzy models build by these methods have a hierarchical structure: each split node of the tree defines one membership function.

The fuzzy tree learning methods are, however, more computationally complex, as compared to the crisp ones, because 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) [26] approach.

Flat vs hierarchical fuzzy model structure. It is worth noting that the fuzzy models used in this paper, produced by the Lolimot method are flat Takagi-Sugeno fuzzy models, in the sense that all normalized membership functions could be calculated in parallel [42]. In particular, the input space partitioning task in Lolimot is performed by a hierarchical tree learning method. On the other hand, the structure of the fuzzy trees as built in the previously mentioned related works

[23], [24], [25], is different: the fuzzy models produced there are hierarchical. There, sigmoidal membership functions corresponding to each split in the tree are applied in a hierarchical fashion, to obtain the final fuzzy membership values. In the remainder of the paper we will use the term *fuzzy linear model tree* or simply *fuzzy tree* to denote a flat TS model, built by Lolimot.

D. Tree ensembles

Ensembles [27]–[29], or committees of predictors, work by creating several *base models*. Each base model, i.e. model tree, is an imperfect predictive model capturing a potentially different aspect of the system being modeled. Combining the imperfect predictions obtained from each base model should improve the predictive power over a single tree and thus provide a more accurate model.

Ensembles based on the bootstrap resampling principle [30] identify each base model from a modified version of the training data. First, several bootstrap samples, i.e., samples with replacement, of equal cardinality to the training data are created from the training data, after which a tree is built for each sample. This procedure, called bagging [30], [31], can be also used for regression tasks. However, most of the tree ensembles for regression consider piecewise constant models, i.e., regression trees.

The more powerful piecewise linear models are rarely used in the ensemble setting, with three exceptions [32]–[34]. The semi-random model tree ensembles of [32] utilize the bagging principle, where the model trees use a non-standard heuristic for the splits, i.e., median splits, for handling outliers. The work of [33] introduces ensembles of crisp model trees, using trees with random splits. The work of [34] introduces ensembles of model trees, where crisp model trees are built in a first stage, and fuzzification of the model trees is performed in a second stage.

Several different approaches of learning ensembles of fuzzy trees have also been introduced, but most of them only for classification tasks [22], [35]–[37]. The ensembles introduced follow the principles of bagging, boosting [37], and keeping several options for a split node, i.e., option trees [17], [36]. The only ensemble of fuzzy model trees approach [38] known to us, builds ensembles using the complete training data set. The trees are induced by means of crisp tree induction first, and are later fuzzified: The parameters of the fuzzy membership functions are optimized.

III. Methodology

This section introduces fuzzy linear model trees for multi-outputs and describes the procedure for building ensembles of these models. It starts by introducing the fuzzy linear model tree, which is able to predict multiple output variables. Then, the iterative procedure for building model trees is described: the tree learning is described first, followed by the estimation of the local models. A summary of the algorithm modifications for multiple-outputs follows. The last subsection gives the outline of the ensemble building algorithm.

In the remainder of the paper we will use the following notation:

D	the data points available for training
$n = \|D\|$	the number of data points in D
r	the number of outputs (targets)
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

A. Fuzzy linear model trees

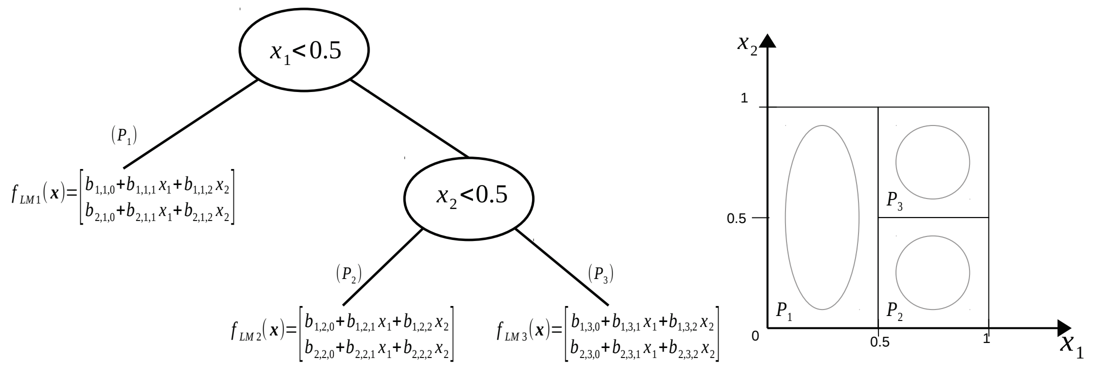
The fuzzy models used in this paper utilize a tree learning procedure [21] for the input space partitioning. During the tree learning, a tree structure composed out of split nodes and terminal nodes is created. The split nodes contain a split of the form *variable* $\leq$ *threshold* and split the input space into two subregions. By following all split nodes from the root down to the a particular terminal node, partitions P_i of the input space can be obtained (Figure 1).

Given identification data $D = \{(\boldsymbol{x}, \boldsymbol{y}) | \boldsymbol{x} \in R^p, \boldsymbol{y} \in R^r\}$, where $\boldsymbol{x}$ is the vector of features (inputs) and $\boldsymbol{y}$ is the vector of outputs, a model tree partitions the input space R^p into several partitions P_i. A local model is estimated for each partition. For example, take the model tree shown on the left-hand side of Figure 1, which defines a function $\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{y}(x_1, x_2) = (y_1(x_1, x_2), y_2(x_1, x_2))$ on the unit square $(x_1, x_2) \in [0, 1] \times [0, 1]$. The three partitions defined by the model tree are:

$$P_1 = \{\boldsymbol{x} \in R^p | 0 \leq x_1 < 0.5\} \tag{1}$$

$$P_2 = \{\boldsymbol{x} \in R^p | 0.5 \leq x_1 \leq 1 \wedge 0 \leq x_2 < 0.5\} \tag{2}$$

$$P_3 = \{\boldsymbol{x} \in R^p | 0.5 \leq x_1 \leq 1 \wedge 0.5 \leq x_2 \leq 1\}. \tag{3}$$



$x_1 < 0.5$
(P_1)
$f_{LM1}(\boldsymbol{x}) = \begin{bmatrix} b_{1,1,0} + b_{1,1,1}x_1 + b_{1,1,2}x_2 \\ b_{2,1,0} + b_{2,1,1}x_1 + b_{2,1,2}x_2 \end{bmatrix}$
$x_2 < 0.5$
(P_2)
(P_3)
$f_{LM2}(\boldsymbol{x}) = \begin{bmatrix} b_{1,2,0} + b_{1,2,1}x_1 + b_{1,2,2}x_2 \\ b_{2,2,0} + b_{2,2,1}x_1 + b_{2,2,2}x_2 \end{bmatrix}$
$f_{LM3}(\boldsymbol{x}) = \begin{bmatrix} b_{1,3,0} + b_{1,3,1}x_1 + b_{1,3,2}x_2 \\ b_{2,3,0} + b_{2,3,1}x_1 + b_{2,3,2}x_2 \end{bmatrix}$
x_2
1
0.5
0
P_1
P_2
P_3
0.5
1
x_1

Figure 1: A multi-output model tree and its corresponding partitioning of the input space. The tree contains 3 local models, and the number of output variables is 2.

The partitioning is complete, as $\cup P_j = R^p$, and the partitions are disjoint subsets, since $P_i \cap P_j = \emptyset$, for $i \neq j$.

After the tree partitioning is finished, the tree structure is transformed into a flat one, by using only the produced partitions. In particular, partition P_j is associated with a multi-dimensional Gaussian fuzzy membership function $\Phi_j(\boldsymbol{x})$. The fuzzy membership function determines the membership of each data point, and is symbolically shown using ellipses in Figure 1. It is used in the calculation of the final output of the fuzzy linear model tree, which has the TS form:

$$\hat{\boldsymbol{y}}(\boldsymbol{x}) = \sum_{j=1}^{m} \Phi_j(\boldsymbol{x}) f_{LMj}(\boldsymbol{x}). \tag{4}$$

The local models f_{LM} can be realized as simpler models (constant, linear, affine) or more complex models (polynomial, rational). In this work, we use local affine models, which we also refer to as local models.

Multi-output TS model. One of the aims of the paper is to analyze multi-output systems. In a typically used baseline approach, r MISO models are built, each for prediction of one output variable. Presuming that the MISO model built by Lolimot for output l is a TS model with m_l rules, the output for output variable l is calculated as:

$$\hat{y}_l(\boldsymbol{x}) = \sum_{j=1}^{m_l} (b_{l,j,0} + b_{l,j,1}x_1 + b_{l,j,2}x_2 + ... + b_{l,j,p}x_p) \Phi_j^{(l)}(\boldsymbol{x}). \tag{5}$$

The local affine model here, for local model j and output l, is determined by the coefficient vector $\boldsymbol{b}_{l,j} = [b_{l,j,0}, b_{l,j,1}, b_{l,j,2}, ...b_{l,j,p}]^T$.

In the MIMO case, Lolimot is used to build a ”multi-output Takagi-Sugeno model”. Each of the partitions resulting from the tree partitioning, is associated with a local models which now has vector form (c.f. Figure 1). This multi-output Takagi-Sugeno model contains m rules, and the prediction for output variable l is calculated as:

$$\hat{y_l}(\boldsymbol{x}) = \sum_{j=1}^{m} (b_{l,j,0} + b_{l,j,1}x_1 + b_{l,j,2}x_2 + ... + b_{l,j,p}x_p)\Phi_j(\boldsymbol{x}). \tag{6}$$

Using this multi-output methodology, the resulting multi-output TS model contains a single common antecedent part, and different consequent parts for each of the outputs.

Advantages of tree partitioning. The task of input space partitioning has been solved using different approaches. Some of the more widely used ones are grid partitioning [19] and product space clustering [39]. This work considers tree partitioning, used in Lolimot, which has certain advantages over the other approaches.

The tree partitioning is able to handle large dimensional problems: it is able to create partitions of different sizes, and in those parts of the input space where finer partitioning is needed. On the other hand, grid partitioning suffers from the curse of dimensionality, as it creates a complete grid over the input space. Although approaches to reduce the complete grid exist [40], they are usually restricted to low-dimensional problems.

The Gustafson-Kessel algorithm [41] is one of the more popular product space clustering approaches. One of its disadvantages is that the number of clusters has to be set upfront, while the tree partitioning procedure can determine it automatically. Also, the computational time required for this product space clustering approach increases fast with the dimensionality, which is not the case with tree partitioning approaches.

Another related work deals with the input space partitioning problem by using a combination of fuzzy clustering, with the Gustafson-Kessel algorithm, and supervised learning [42]. The method named SuHiClust performs a more complicated type of tree partitioning, than the one presented here: it is able to produce axes-oblique splits and splits with arbitrary proportion (not just half-half). However, this type of partitioning requires more training time than the one presented here.

B. *Learning the tree structure*

The learning of the tree structure, defined by the hierarchy of split nodes, terminal nodes, and local models, in Lolimot [8] is performed by an iterative procedure. 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.

```
1  Algorithm Learn_ensemble($D$)
       Data: data set $D$
       Result: an ensemble $E$
2      Create $t$ bootstrap samples of $D$ : $D_1, D_2, .., D_t$
3      Build a tree using each of the $t$ samples: $T_i$ = Build_Lolimot_tree($D_i$)
4      Return the ensemble $E = \{T_1, T_2, .., T_t\}$
5  Algorithm Build_Lolimot_tree($D_{learn}$, $D_{sim}$)
       Data: data set used for learning $D_{learn}$, data set used for simulation $D_{sim}$
       Result: model tree $T$
6      Create a root node for the tree and estimate local model parameters using data $D_{learn}$
7      Evaluate the model tree: ($e$, $t$) = Evaluate($T$, $D_{learn}$, $D_{sim}$)
8      while the maximal number of LMs is not reached do
9          Select the terminal node $t$ for splitting
10         Create the set $S$ of candidate splits using terminal node $t$
11         for all candidate splits $s$ in $S$ do
12             Replace the terminal node $t$ with a split node $t'_s$ and split $s$
13             Create two terminal nodes $t'_1$ and $t'_2$ as descendants of $t'_s$
14             Estimate local model parameters for $t'_1$ and $t'_2$ by weighted linear regression using $D_{learn}$
15             Evaluate the model tree: ($e$, $t$) = Evaluate($T$, $D_{learn}$, $D_{sim}$)
16         end
17         Select and keep the candidate split $s$ which produces lowest overall error $e_s$
18     end
19     Let $i_{AIC}$ be the iteration with the smallest AIC value
20     Return the model tree $T$ from iteration $i_{AIC}$
21 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
22     if $D_{learn}$ are data from a dynamic system then
23         Perform simulation of $T$ using the data points $D_{sim}$
24         Let $e$ be the output error
25     else
26         Let $e$ be the prediction error of $T$ calculated using $D_{learn}$
27     end
28     Let $t$ be the terminal node of $T$ corresponding to largest sum of squared errors
29     Return ($e$, $t$)
```

Algorithm 1: Pseudocode for the model tree ensemble learning method.

For the purpose of describing the method, we define a *candidate split* s to be a triple consisting of a terminal node of the tree built so far, an input variable to split on, and a split threshold. In

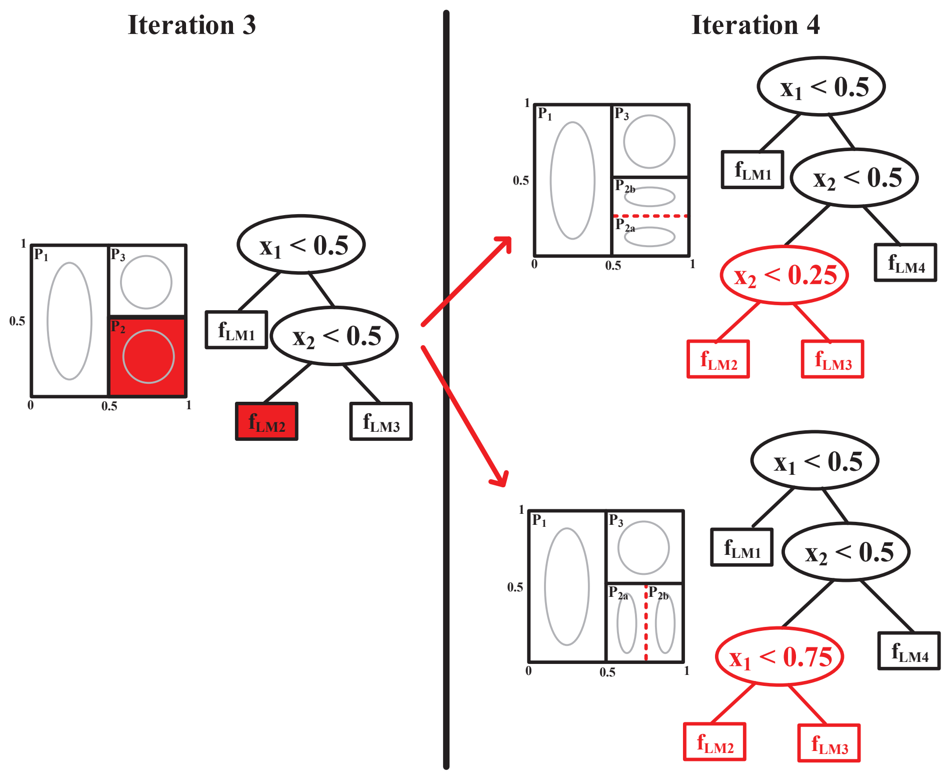


Figure 2: Two iterations of Lolimot. The left-hand side depicts the tree structure and its partitioning in the third iteration. The marked terminal node is the one selected for further splitting. The right-hand side depicts the complete set of candidate splits S, considered in the fourth iteration.

each iteration, several possible candidate splits are considered. The set of candidate splits S is created using the following procedure: one terminal node is selected for further splitting from the existing tree and half-splits, i.e., splits that divide the current partition into two equal halves, in all dimensions are considered as candidate splits.

Each of the candidate splits is evaluated by using a heuristic greedy evaluation function: (a) a tree with the candidate split is created, (b) the local models are estimated for the added terminal nodes, and (c) the fit 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 2.

Selection heuristic. The Lolimot algorithm considers only one existing terminal node for expansion. It selects the terminal node which gives rise to the largest sum of squared errors. In each iteration, the model built so far is evaluated either using simulation or prediction, c.f., the function Evaluate in the pseudocode shown in Algorithm 1. 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.

We can use 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. We can also use a selection heuristic which utilizes (one-step-ahead) prediction.

C. Estimation of local models parameters

In each iteration of the method, 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 of the partitions and the corresponding local models. The Lolimot method uses the multi-dimensional Gaussian membership function [8], whose center **c** is determined by the center of the partition, and standard deviation vector $\boldsymbol{\sigma}$ is calculated as $1/3$ of the size of the partition [3]. For example, for partition P_1 of the tree in Figure 1, $\mathbf{c} = [0.25, 0.5]^T$ and $\boldsymbol{\sigma} = [0.5/3, 1/3]^T$.

The membership of a data point **x** to the j-th partition is calculated as

$$\mu_j(\mathbf{x}) = exp(-\frac{1}{2}\sum_{i=1}^{n}(\frac{x_i - c_i}{\sigma_i})^2). \tag{7}$$

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

$$\Phi_j(\mathbf{x}) = \frac{\mu_j(\mathbf{x})}{\sum_{k=1}^{m}\mu_k(\mathbf{x})} \tag{8}$$

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

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 multiple-output case, this problem can be formulated as an optimization problem, where the objective function is:

$$I = \sum_{l=1}^{r} \sum_{i=1}^{n} e_{i,l}^2 \tag{9}$$

and $e_{i,l}^2 = (y_{i,l} - \hat{y}_{i,l})^2$ for output variable l and data point i.

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

$$I_l = \sum_{i=1}^{n} e_{i,l}^2 \quad l = 1, 2, .., r. \tag{10}$$

The parameter vector for output variable l contains $M \cdot (p+1)$ parameters

$$\boldsymbol{b}_l = \begin{bmatrix} b_{l,1,0} & b_{l,1,1} & \cdots & b_{l,1,p} & \cdots & b_{l,m,0} & b_{l,m,1} \cdots & b_{l,m,p} \end{bmatrix}^T . \tag{11}$$

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

In the 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 [43] estimates the parameters of each local model in isolation of each other. An alternative is the *global estimation* procedure [19], which estimates all of the local model parameters simultaneously. The former one is faster, more stable, shows better performance in noisy situations, and allows for a better interpretability of consequents [44], [45] but doesn't take the interactions between local models, as defined by the membership function, into account.

As previously discussed, the parameter estimation for each output is treated as a separate problem. So, the local parameter estimation for all r outputs requires $r \cdot m$ separate weighted least square regressions. For output l and local model j, the regression matrix $\boldsymbol{X}_{l,j}$ is a $n \times (p+1)$ matrix, containing the values of the regressor variables

$$\boldsymbol{X}_{l,j} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,p} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \cdots & x_{n,p} \end{bmatrix} \tag{12}$$

The weighting matrix $Q_{l,j}$, composed of the validity function values (Eq. 8) is :

$$Q_{l,j} = \begin{bmatrix} \Phi_j(\mathbf{x}_1) & 0 & \cdots & 0 \\ 0 & \Phi_j(\mathbf{x}_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Phi_j(\mathbf{x}_n) \end{bmatrix}. \tag{13}$$

We define the vector of outputs for the output variable l, $\mathbf{y}_l$ as

$$\mathbf{y}_l = \begin{bmatrix} y_{l,1} & y_{l,2} & \cdots & y_{l,n} \end{bmatrix}^T, \tag{14}$$

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

$$\hat{\boldsymbol{b}}_{l,j} = (\boldsymbol{X}_{l,j}^T Q_{l,j} \boldsymbol{X}_{l,j})^{-1} \boldsymbol{X}_{l,j}^T Q_{l,j} \boldsymbol{y}_l. \tag{15}$$

D. Multiple-outputs

This section summarizes the modifications of the model tree learning algorithm that are necessary for handling multiple-outputs. These modifications are empirically evaluated in the next section. There are three modifications for multiple-outputs, as compared to the single-output version. These are:

- **Selecting the partition for further splitting.** The algorithm chooses only one partition (terminal node of the tree) for further splitting. The criterion to choose (line 9 in Algorithm 1) is the sum of squared errors of the data points. While in the SO case this is trivial, in the MO case, the squared errors for each output are summed, without output weighting. The error of partition P_j is calculated as:

$$e^2(P_j) = \sum_{i=1}^{n} \sum_{l=1}^{r} (y_{i,l} - \hat{y}_{i,l})^2 \Phi_j(x_i). \tag{16}$$

  The partition which results in the largest error, i.e., sum of squared errors, is selected for further splitting.
- **Estimation of local model parameters.** The estimation of local model parameters, as shown in Eq. 15, is performed separately for each output variable. This means that it is treated as a separate optimization problem for each output.
- **Cost function for the multi-output case.** After the two new local models (corresponding to a candidate split) are estimated, the whole model is evaluated, as shown in line 15

of Algorithm 1. In the multi-output case, the cost function aggregates the errors of each output variable. We use the multi-output root relative mean-squared error (RRSE) error, also denoted as normalized root mean squared error [8], which is calculated as:

$$RRSE = \sqrt{\frac{\sum_{l=1}^{r} \sum_{i=1}^{n} (\hat{y}_{i,l} - y_{i,l})^2}{\sum_{l=1}^{r} \sum_{i=1}^{n} (\bar{y}_l - y_{i,l})^2}}. \tag{17}$$

E. Ensemble creation

The model tree ensembles are built by using bagging. Bootstrap replicates are created, i.e., random samples with replacement of the training data set D that have an equal number of data points as the training set. Each of the replicates D_i is used to build one fuzzy model tree. As shown in the pseudocode function $Learn\_ensemble(D)$ in Algorithm 1, the bootstrap sample D_i is used as a data set D_{learn} for learning a Lolimot model tree.

The learning procedure starts by creating t bootstrap replicates of the training data. Using each of the t data samples, a collection of fuzzy linear model trees is built: $f_1, f_2, .., f_t$. Denoting the predictions of the t multi-output model trees of the ensemble with $\hat{f}_i(\mathbf{x})$, the overall prediction from the model tree ensemble is the average of the base model predictions for one output variable:

$$\hat{y}(\mathbf{x}) = \frac{1}{t} \sum_{i=1}^{t} \hat{f}_i(\mathbf{x}). \tag{18}$$

For the case of modeling dynamic systems, the Lolimot base learning algorithm uses a split selection (model evaluation) heuristic which performs simulation. As the requirement for simulation is to have time contiguous data points as input, the whole training sample is used to perform the task of simulation. This is also evident from the pseudocode for the algorithm, given in Algorithm 1.

IV. Experimental evaluation setup

This section describes the setup for the evaluation of the ensembles of model trees for multi-output dynamic system identification. Note that we can use the ensembles in two different modes, defined by the search heuristic employed to learn the individual trees. Namely, we can either use the default simulation/output error (Lolimot$_s$ and Bagging$_s$), or prediction error (Lolimot$_p$ and Bagging$_p$) as a search heuristic.

In the remainder, the experimental setup is described first, followed by the order selection procedure and the parameters of the methods. This is followed by a description of the six multiple-output case studies.

A. Experimental setup

The experimental setup uses input-output data for each of the case studies, that are split into two sets: a training set and a testing set. The model is built by using the training set and evaluated by using the testing set. The error reported is in terms of root relative mean-squared error (RRSE). In the comparisons, we report the aggregated error of the model over all r output variables, calculated according to Equation 17.

The evaluation compares r single-output (SO) models, built for predicting one output variable each, to a single multi-output (MO) model, which predicts all output variables simultaneously. We use two different types of single-output models: Lolimot model trees (Lolimot$_s$(SO) and Lolimot$_p$(SO)) and ensembles thereof (Bagging$_s$(SO) and Bagging$_p$(SO)). We also consider two types of multi-output models: multi-output Lolimot trees (Lolimot$_s$(MO) and Lolimot$_p$(MO)) and bagging thereof (Bagging$_s$(MO) and Bagging$_p$(MO)).

The models are evaluated in terms of output error, by using the procedure of simulation. This presents a more control engineering oriented and stringent type of evaluation of the model, given the realistic possibility of error accumulation. The performance of the SO models is evaluated using **parallel simulation**: The predicted values of each of the SO models at time point k are used in the regressor vector at time point $k+1$. The simulation procedure of a MO model uses the predicted values by the MO model at time point k for each output in the regressor vector at time point $k+1$.

To asses the relative performance of all the algorithm variants considered across all the datasets, we follow the methodology suggested in [46]. We perform the nonparametric Friedman test, along with the Nemenyi post-hoc test, to look for statistically significant differences in the predictive performance of the method variants. Their outcome is depicted by an average ranking diagram, where methods to the right perform better and method pairs whose performance difference is less than the critical distance (CD) do not differ significantly.

B. Order and regressor selection

This subsection describes the necessary transformations of the input-output data, performed in order to successfully model the dynamic system using a Lolimot tree or an ensemble of Lolimot trees. The data transformation is performed according to the external dynamics approach [3], which requires that a value for the order (lag) is selected.

A heuristic procedure was used, i.e. a forward selection procedure. It was utilized to determine (a) the optimal order (lag) for each case study and (b) the optimal set of regressors for each output. The procedure started from an empty set of regressors for each output, and in each step added the regressor which helped to obtain the lowest error. The criterion used to decide which regressor to add was the average squared error of single- and multi-output model trees. The error was calculated on the validation set, i.e. a small subset of all the training data available for the case study. The forward selection procedure was however limited: it considered only certain orders, i.e. only up to a certain number of previous values for a variable, as shown in Table I. The selected regressors by the forward selection procedure are reported in Table II.

Table I: The considered and selected model orders by using the forward selection procedure, and the datasets created.

Case study	**Orders considered**	**Order selected**	**Dataset(s)**
Gas-liquid separator	1,2,3	2	GLS
Winding	1,2,3	2	W
Cascaded tanks	1	2	CT, CT$_N$
Steam gen.	3,4,5,6,7	2	SG
Cont.stirred tank reactor	1,2,3,4	2	CSTR, CSTR$_N$
Robot arm	1,2	2	RA

The same regressors were then used for all single tree and ensemble methods. Note that the data of four case studies were already disturbed by noise, while for CSTR and CT we added noise in the output variables, thus producing a total of eight datasets (Table I).

C. Parameters of the method

The parameter settings of the model tree ensembles are described next. The parameters of the Lolimot model tree algorithm are discussed first, followed by a discussion of the parameters of the ensemble method.

1) Parameters of the model tree algorithm: The single parameter that is set for the model tree algorithm is the maximal size of the tree, which effectively performs tree pre-pruning. The model tree algorithm stores the tree in each iteration and suggests a tree with an optimal size, i.e., number of local models. For efficiency reasons, we use a value of 15 for this parameter.

Table II: The regressors selected by using the forward selection procedure.

Dataset	Output variable	Regressors selected
GLS	$p_1(k)$	$u_1(k-1)$, $u_2(k-2)$, $p_1(k-1)$, $h_1(k-1)$
	$h_1(k)$	$u_1(k-2)$, $u_2(k-1)$, $h_1(k-1)$
W	$T_1(k)$	$S_2(k-2)$, $S_2(k-1)$, $S_3(k-2)$, $M_1(k-2)$, $M_1(k-1)$, $M_2(k-2)$, $M_2(k-1)$, $T1(k-2)$, $T1(k-1)$, $T_2(k-2)$, $T_2(k-1)$
	$T_2(k)$	$S_1(k-2)$, $S_2(k-2)$, $S_2(k-1)$, $S_3(k-2)$, $M_1(k-2)$, $M_2(k-2)$, $T_2(k-2)$, $T_2(k-1)$
CT, CT$_N$	$h_1(k)$	$h_3(k-2)$, $h_4(k-2)$, $h_1(k-1)$
	$h_2(k)$	$q1(k-2)$, $q1(k-1)$, $h_4(k-2)$, $h_2(k-2)$, $h_2(k-1)$
	$h_3(k)$	$q1(k-2)$, $q1(k-1)$, $h_3(k-1)$, $h_4(k-2)$
	$h_4(k)$	$q1(k-2)$, $q1(k-1)$, $q2(k-1)$, $h_4(k-1)$
SG	$y_1(k)$	$u_1(k-2)$, $u_1(k-1)$, $u_3(k-1)$, $u_4(k-2)$, $u_4(k-1)$, $y_1(k-2)$, $y_1(k-1)$, $y_3(k-1)$, $y_4(k-2)$, $y_4(k-1)$
	$y_2(k)$	$u_2(k-2)$, $u_2(k-1)$, $y_1(k-2)$, $y_2(k-1)$
	$y_3(k)$	$u_3(k-2)$, $u_3(k-1)$, $u_4(k-2)$, $y_2(k-2)$, $y_3(k-2)$, $y_3(k-1)$, $y_4(k-2)$, $y_4(k-1)$
	$y_4(k)$	$u_1(k-1)$, $u_3(k-1)$, $u_4(k-2)$, $u_4(k-1)$, $y_1(k-1)$, $y_3(k-1)$, $y_4(k-2)$, $y_4(k-1)$
CSTR, CSTR$_N$	$Ca(k)$	$q(k-1)$, $Ca(k-1)$, $T(k-1)$
	$T(k)$	$q(k-2)$, $q(k-1)$, $Ca(k-2)$, $Ca(k-1)$, $T(k-1)$
RA	$t_1(k)$	$x_1(k-1)$, $x_2(k-1)$, $x_4(k-1)$, $x_7(k-2)$, $x_7(k-1)$, $v_1(k-2)$, $v_2(k-2)$, $v_3(k-2)$, $v_4(k-2)$, $v_4(k-1)$, $v_5(k-2)$, $v_5(k-1)$, $v_6(k-1)$, $v_7(k-2)$, $v_7(k-1)$, $a_1(k-2)$, $a_1(k-1)$, $a_3(k-1)$, $a_4(k-2)$, $a_4(k-1)$, $a_5(k-2)$, $a_5(k-1)$, $a_7(k-2)$, $a_7(k-1)$, $t_1(k-1)$, $t_2(k-1)$, $t_5(k-1)$
	$t_2(k)$	$x_1(k-1)$, $x_2(k-2)$, $x_3(k-2)$, $x_4(k-1)$, $x_5(k-2)$, $x_5(k-1)$, $x_6(k-2)$, $x_7(k-1)$, $v_1(k-2)$, $v_2(k-2)$, $v_3(k-2)$, $v_4(k-1)$, $v_5(k-2)$, $v_5(k-1)$, $v_6(k-1)$, $v_7(k-2)$, $v_7(k-1)$, $a_4(k-1)$, $a_5(k-2)$, $a_5(k-1)$, $a_6(k-2)$, $a_6(k-1)$, $t_1(k-2)$, $t_1(k-1)$, $t_2(k-2)$, $t_2(k-1)$, $t_3(k-1)$, $t_6(k-2)$
	$t_3(k)$	$x_1(k-1)$, $x_2(k-1)$, $x_3(k-2)$, $x_4(k-1)$, $x_5(k-2)$, $x_6(k-1)$, $x_7(k-2)$, $v_1(k-2)$, $v_1(k-1)$, $v_3(k-1)$, $v_4(k-2)$, $v_4(k-1)$, $a_1(k-2)$, $a_2(k-2)$, $a_2(k-1)$, $a_3(k-2)$, $a_3(k-1)$, $a_4(k-2)$, $a_4(k-1)$, $a_5(k-2)$, $a_5(k-1)$, $a_6(k-1)$, $a_7(k-2)$, $a_7(k-1)$, $t_1(k-1)$, $t_2(k-2)$, $t_3(k-1)$
	$t_4(k)$	$x_6(k-2)$, $x_6(k-1)$, $v_1(k-2)$, $v_2(k-1)$, $v_3(k-2)$, $v_4(k-2)$, $v_5(k-2)$, $v_5(k-1)$, $v_6(k-2)$, $v_7(k-2)$, $a_5(k-2)$, $a_7(k-2)$, $t_3(k-1)$, $t_4(k-1)$, $t_7(k-1)$
	$t_5(k)$	$x_2(k-1)$, $x_3(k-1)$, $x_4(k-1)$, $v_1(k-2)$, $v_7(k-1)$, $a_2(k-1)$, $a_6(k-1)$, $t_2(k-1)$, $t_4(k-1)$, $t_5(k-2)$, $t_5(k-1)$, $t_6(k-2)$
	$t_6(k)$	$x_1(k-2)$, $x_1(k-1)$, $x_2(k-2)$, $x_6(k-2)$, $x_6(k-1)$, $x_7(k-1)$, $v_1(k-1)$, $v_2(k-1)$, $v_4(k-2)$, $v_6(k-2)$, $a_1(k-2)$, $a_4(k-2)$, $a_4(k-1)$, $a_5(k-2)$, $a_5(k-1)$, $a_7(k-2)$, $t_1(k-2)$, $t_3(k-1)$, $t_5(k-1)$, $t_6(k-1)$, $t_7(k-1)$
	$t_7(k)$	$x_4(k-1)$, $v_1(k-2)$, $v_3(k-1)$, $v_5(k-1)$, $v_7(k-1)$, $a_2(k-2)$, $t_4(k-2)$, $t_4(k-1)$, $t_7(k-2)$, $t_7(k-1)$

Additionally, the optimal size of a Lolimot tree is determined using the Akaike Information Criterion (AIC). It takes into consideration the size of the training set n, the error of the model for each iteration and the number of coefficients in the linear models:

$$AIC = n \cdot log(e_{MSE}) + r \cdot m \cdot (p+1), \tag{19}$$

where e_{MSE} denotes the mean-squared error of the single- or multi-output model. The aim of AIC is to give a good bias/variance trade off, and the purpose of bagging is to reduce variance [47]. Bagging could use Lolimot models built with AIC termination or without: all ensemble results in our paper are produced by using Lolimot with AIC termination [1].

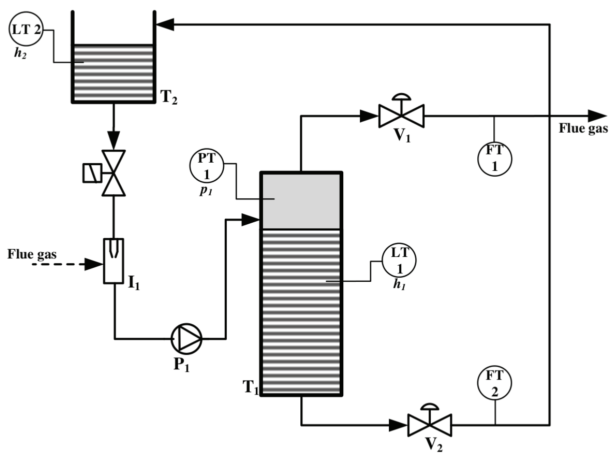
2) Parameters of the ensemble algorithm: We vary the number of trees in the ensemble, considering up to 50 trees. The results for all of these are shown in graphical form in Figure 10. Based on these, we can examine the influence of this parameter on performance and select an appropriate value for it.

[1]Additional experiments of bagging Lolimot models without AIC showed identical conclusions.

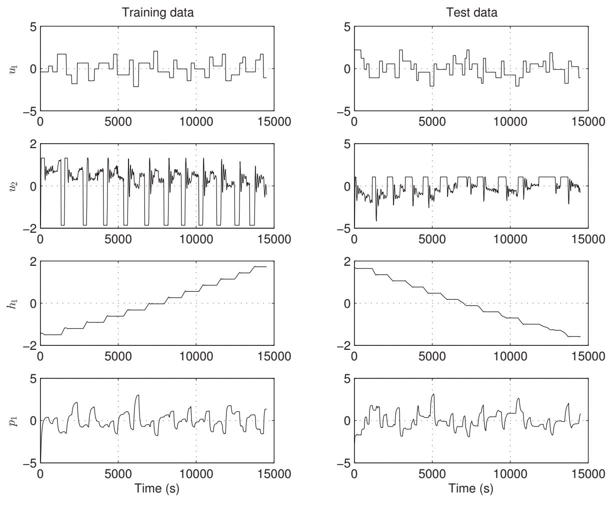
D. Case Study: Gas-liquid Separator

The system being modeled in this case study is a unit for the separation of gas from liquid [48]. 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 3 (a).

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.



(a)



(b)

Figure 3: (a) A schematic diagram of the semi-industrial process plant. (b) Detrended data for the identification of the GLS system.

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:

$$\begin{aligned} \frac{dp_1}{dt} &= f_a(h_1)[\alpha_1 + \alpha_2 p_1 + \alpha_3 p_1^2 + f_b(u_1)\sqrt{p_1} + \\ &\quad + f_c(u_2)\sqrt{(p_1 + \alpha_4 + \alpha_5 h_1)}] \end{aligned} \tag{20}$$

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:

$$\frac{dh_1}{dt} = \alpha_6 + f_c(u_2)\sqrt{(p_1 + \alpha_4 + \alpha_5 h_1)} \tag{21}$$

where the value α_6 is a constant [48].

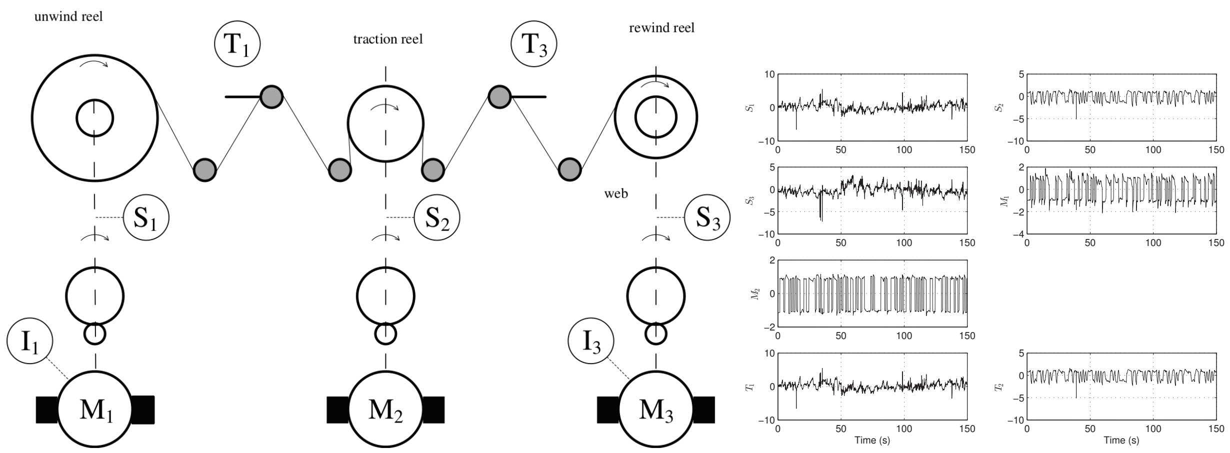
The aim of the system identification in this case study is to build a model for predicting the variables p_1 and h_1, from their lagged values, as well as lagged values of the input variables. The sampling time selected was 20 s [49]. The training and the test data both consist of 733 input-output data points, shown in Figure 3 (b), and are disturbed by intrinsic measurement noise.

E. Case Study: Winding Process

The winding process is a setup of an industrial winding process pilot plant [50]. 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 4 (a).

The task is to control the web tension in order to avoid sliding effects, wrinkles and material distortion. It presents a nonlinear and time-varying 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.

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 [51]. The sampling rate is 0.1 s. There are 2500 data points, covering 250 s. The input-output data used are shown in Figure 4 (b).



(a) (b)

Figure 4: (a) A diagram of the industrial winding process. (b) The available training data.

F. Case study: Cascaded tanks

The case study consists of four cascaded tanks [52] as shown in Figure 5. Liquid flows in the system with flow rates Q_1 and Q_2. These two variables are the inputs. The levels of the liquid in the four tanks, $[h_1, h_2, h_3, h_4]$, are the outputs of this MIMO system.

The cascaded tanks system is described by the nonlinear differential equations: $\dot{\mathbf{h}} = \mathbf{A}\sqrt{\mathbf{h}} + \mathbf{B}\mathbf{Q}$, where

$$\mathbf{A} = \begin{bmatrix} -\frac{s_{2,1}}{s_{1,1}}\sqrt{2g} & 0 & r_{3,1}\frac{s_{2,1}}{s_{1,1}}\sqrt{2g} & r_{3,1}\frac{s_{2,1}}{s_{1,1}}\sqrt{2g} \\ 0 & -\frac{s_{2,1}}{s_{1,1}}\sqrt{2g} & r_{3,1}\frac{s_{2,1}}{s_{1,1}}\sqrt{2g} & r_{3,1}\frac{s_{2,1}}{s_{1,1}}\sqrt{2g} \\ 0 & 0 & r_{3,1}\frac{s_{2,1}}{s_{1,1}}\sqrt{2g} & 0 \\ 0 & 0 & 0 & r_{3,1}\frac{s_{2,1}}{s_{1,1}}\sqrt{2g} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{s_{2,1}}{s_{1,1}} & 0 \\ 0 & \frac{s_{2,1}}{s_{1,1}} \end{bmatrix}.$$

The $\mathbf{h}$ vector denotes the liquid levels, S_1 is the area of the tank ($10^{-3}m^2$), S_2 is the area of the outlet ($10^{-6}m^2$), $r_{i,j}$ is the restriction parameter from tank i to tank j ($r_{3,1}$=$r_{4,2}$ = 0.8), $\mathbf{Q}$ is the inlet flow rate, and g is the gravity constant.

In the original paper [52] where this system is described, the authors use some background knowledge when building the model. In more detail, they determine which regressors are to be used for each of the four outputs based on the design of the system itself. In our work, we instead rely on the results of the regressor selection procedure and use other regressors.

The sampling time is 10 s and there are 1000 simulated data points [53] available (700 for training, 300 for testing) shown in Figure 5 (b). This Figure also shows, that the data contains no noise. We also added noise to the data, to obtain insights into the noise-tolerance of the methods. We added white noise with mean zero and standard deviation of 5% of the standard deviation of the output variables. The noise was added only to the output variables in the training set, and hence to all corresponding lagged variables in the set of features. No noise was added to the test data.

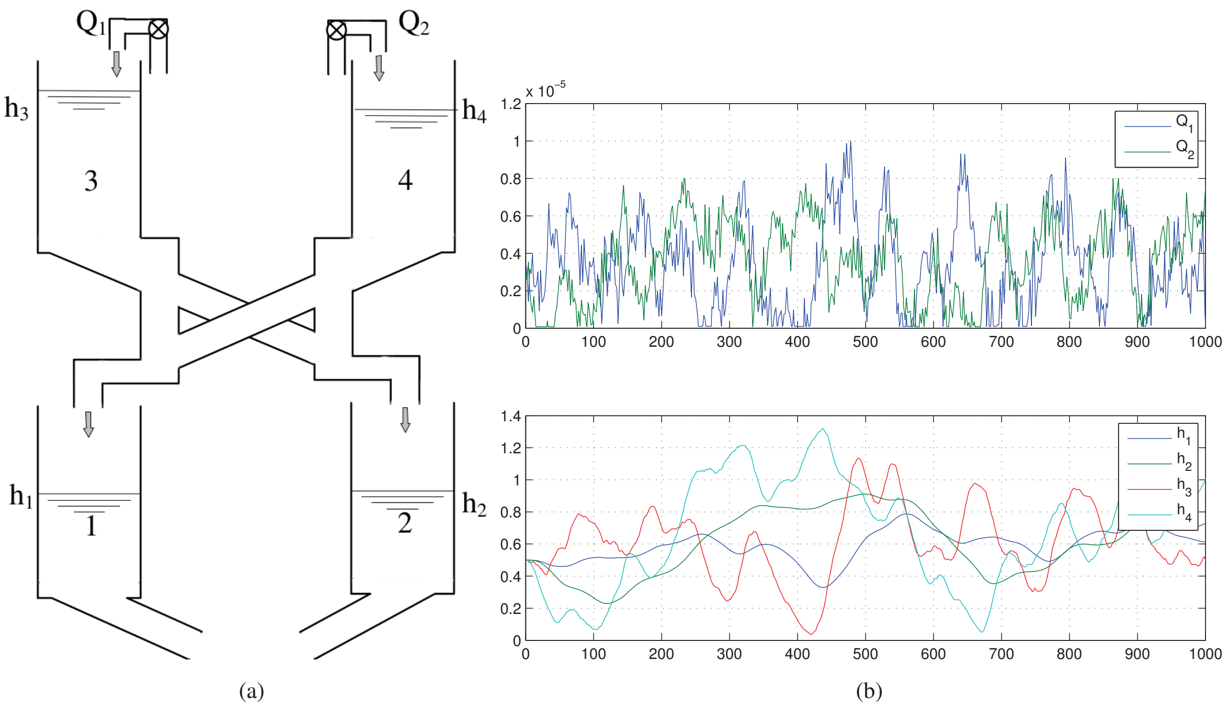


Figure 5: (a) The four cascaded tanks, and (b) the available data (no added noise).

G. *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 [51]. 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. Before identification is performed, the water level control is stabilized using feed-forward and PID control [54]. 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 control objective of preserving the level of the header pressure and the oxygen level in the flue gas [55].

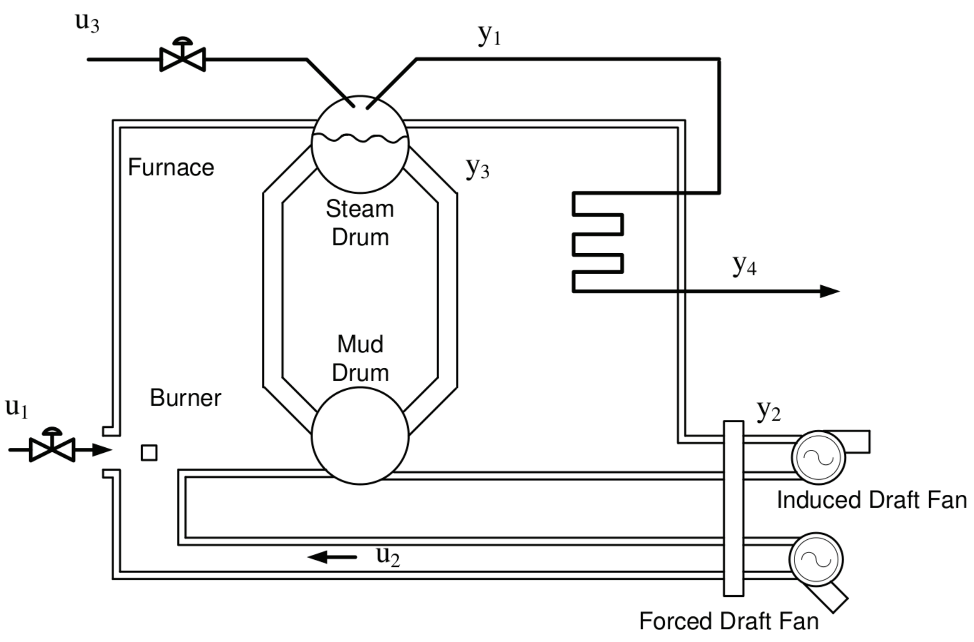


Figure 6: A diagram of the steam generator plant.

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 is depicted in Figure 7. The total number of data points is 9600, out of which the first 7600 were used for training and 2000 were used for testing. The sampling rate is 3 s. 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.

H. Case study: Continuous-stirred tank reactor

This case study concerns the well-known continuous-stirred tank reactor (CSTR). The CSTR process [54] [56] 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

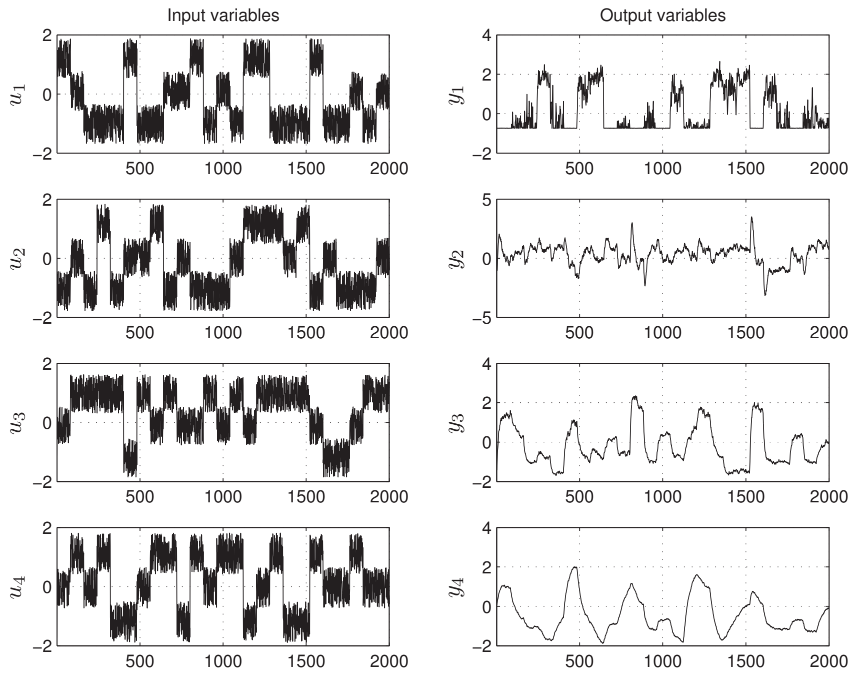


Figure 7: Input-output data of the steam generator dynamic system, used for testing.

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

$$\dot{C}_a(t) = \frac{q}{v}(C_{a0} - C_a(t)) - k_0 C_a(t) e^{-\frac{E}{RT(t)}} \tag{22}$$

$$\begin{aligned} \dot{T}(t) &= \frac{q}{v}(T_0 - T(t)) - k_1 C_a(t) e^{-\frac{E}{RT(t)}} \\ &\quad + k_2 q_c(t)(1 - e^{-\frac{k_3}{q_c(t)}})(T_{c0} - T(t)) \end{aligned} \tag{23}$$

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 available in [56]. The data are obtained from the Daisy repository [51], 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 for training, and the last 2500 for testing [54]. The input-output data of the test set are depicted in Figure 8.

As in the cascaded tanks case study, we add noise to the output variables in the training set. The noise added here has a standard deviation of 20% of the standard deviation of the output variables.

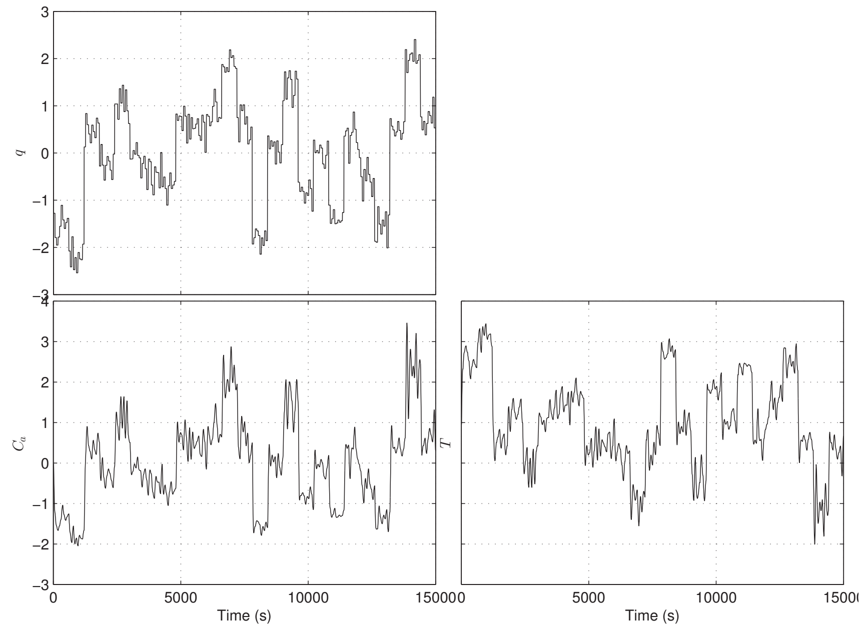


Figure 8: Normalized input-output data of the CSTR dynamic system. Data used for testing.

I. Case study: Robot arm

This case study deals with modeling a 7-degree-of-freedom anthropomorphic robot arm [57]. The data come 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 and 7 output variables. The 21 input dimensions are the 7 joint positions (x_i), 7 joint velocities (v_i), and 7 joint accelerations (a_i), while the 7 output dimensions are the torque commands (t_i) for each motor. The modeling goal is to approximate the torque commands from the input variables.

The number of data points 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 9. The data is nonlinear and contains a small amount of noise.

V. Results

This section presents the results of the performance evaluation of ensembles (bagging) of single- and multi-output model trees, as compared to a set of r single-output trees and a single multi-output tree respectively. The performance is evaluated in terms of the error of the learned

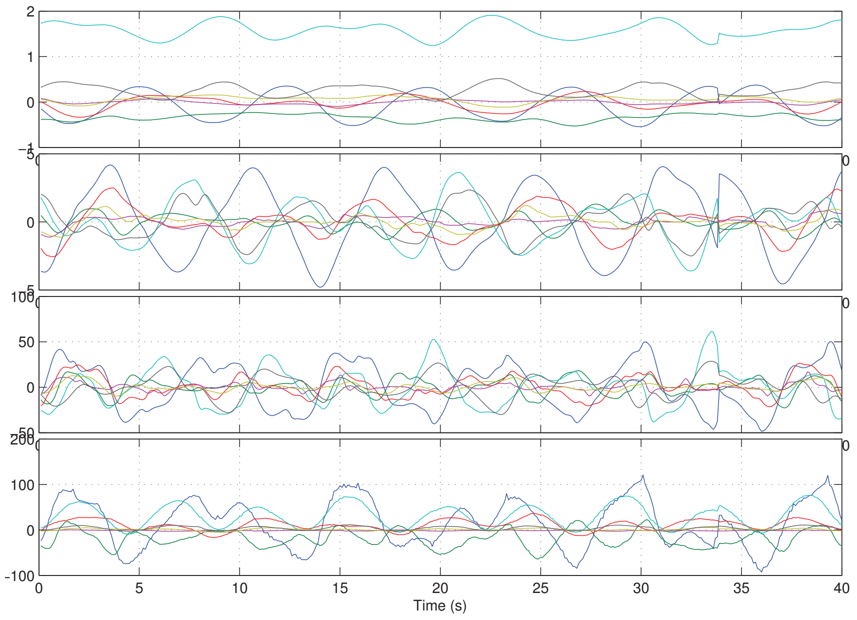


Figure 9: Input-output data for the robot arm system: The first 400 training data points. The figure depicts the 7 joint positions, velocities, accelerations, and torque commands.

model on the testing data for each of the datasets. To learn the individual trees and the ensembles, two selection heuristics are used: output (simulation) error and prediction error.

Figure 10 (a) presents the performance of the models (measured in terms of simulation error) learned by using the simulation error as a search heuristic. Figure 10 (b) presents the performance of the models learned by using one-step-ahead prediction error as a search heuristic. In each of the figures, the left-hand column of plots shows the performance of r SO trees (horizontal line) and r SO tree ensembles (crosses), while the right-hand column shows the performance of the MO tree (horizontal line) and MO tree ensembles (crosses). The y axis depicts the performance measure (RRSE), while the x denotes the number of trees in the ensemble. Note that there are 5 crosses for each number of trees, as 5 runs of bagging with different random seeds were performed to illustrate the effect of the randomness in the bagging procedure.

The results in the left columns of Figure 10 show that bagging of SO model trees improves the performance over a set of SO model trees. The improvements are largest for the W dataset. The results in the right columns show that the multi-output tree ensembles improve the output error over one multi-output tree. For the W dataset, there is clear improvement in performance over a single tree when using an ensemble. Also, a low number of trees in the ensemble is sufficient, both in the SO and MO cases. The figure shows that as few as 20 trees are enough to obtain an improvement over the set of r single-output trees and one multi-output tree, respectively.

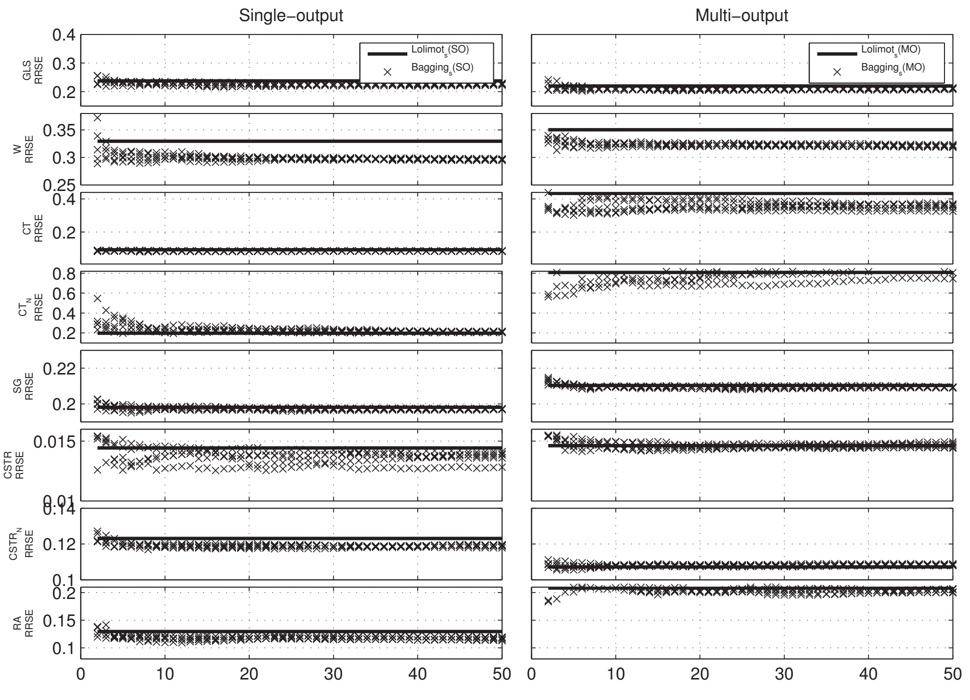
The comparison of the left and the right columns of Figure 10 (b) indicates that the multi-output ensembles have similar performance than that of the single-output ensembles. For the noisy CSTR$_N$ dataset, the multi-output ensemble shows more consistent performance as compared to the single-output ensemble, as the variance of the error is much smaller. For this dataset learning a model either as a single tree or an ensemble for both outputs simultaneously slightly improves overall performance.

The performances of one MO tree and a set of SO trees do not show clear differences, as evident in Figure 10 (b). In the case of the CSTR$_N$ models, the improvement of using a MO tree is clearly visible. However, for the remainder of the datasets, the SO trees show either better of similar predictive performance, as compared to a MO tree.

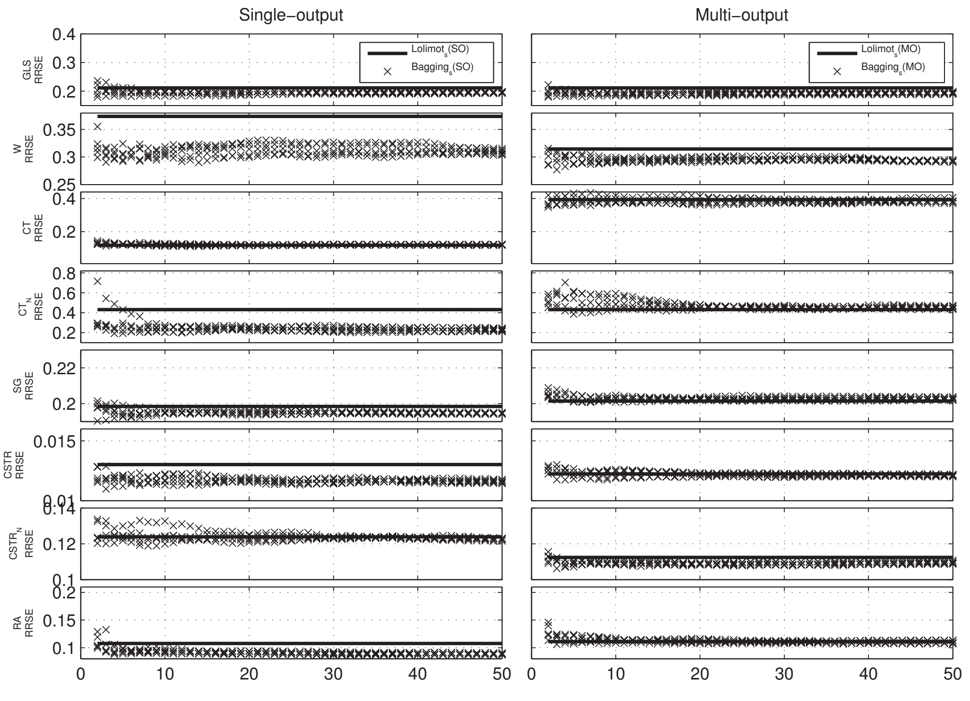
A clearer picture is given by the overall comparison resulting from the Friedman and Nemenyi tests, depicted by the average ranking diagram in Figure 11. The diagram shows the ranking of the method variants and the critical distance. The critical distance is large, which suggests that the differences in performance are not statistically significant for most of the method variants.

The relative performance of the MO approaches and their SO counterparts is clearly visible in Figure 11. The MO approaches are ranked slightly worse than (typically positioned immediately left of) their SO counterparts. The only exception are Lolimot$_s$(SO) and Lolimot$_s$(MO) where the order is reversed. However, the differences are not statistically significant. The ensemble approaches are all on the right and the single tree approaches on the left, clearly showing that the ensemble approaches perform better. In particular, each ensemble method is better ranked than its base method, even if slightly.

Regarding the selection heuristic Figure 11 shows that using simulation rather than prediction error during the construction of the model tree improves the output error of the model. This holds for all corresponding pairs of method variants. The difference in the ranking is largest when comparing the performance of bagging single-output Lolimot trees (however, it is not statistically significant). The benefit of using the simulation error heuristic is pronounced for the noisy CSTR$_N$ and RA datasets. When prediction error is used as a heuristic on these datasets, the error of the single-output tree ensembles is much higher as compared to using simulation error.



(a)



(b)

Figure 10: The performance (simulation error) of the single-output and multi-output trees and ensembles on the testing set. The selection heuristic optimizes (a) the simulation error and (b) the one-step-ahead prediction error on the training set. Missing crosses or horizontal lines indicate a result outside the selected y-axis range (cf. Table III).

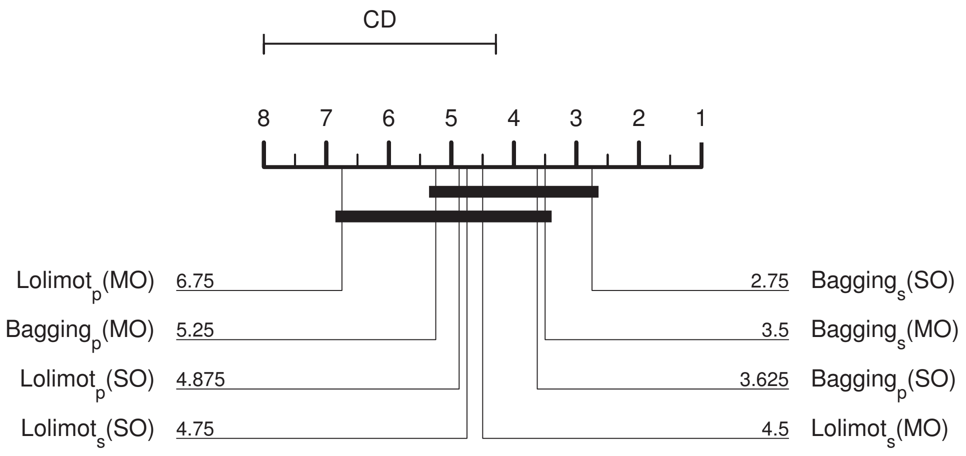


Figure 11: The average rank diagram of the 8 methods on all 8 datasets. Ensembles contain 25 trees. The critical distance is calculated by using a significance level of 5%.

Table III: RRSE values used for the average rank diagram.

	Lol$_s$(SO)	Lol$_s$(MO)	Bagg$_s$(MO)	Bagg$_s$(SO)	Lol$_p$(SO)	Lol$_p$(MO)	Bagg$_p$(MO)	Bagg$_p$(SO)
GLS	0.211638	0.211658	0.191611	0.197134	0.237797	0.219840	0.205417	0.221148
W	0.373771	0.314431	0.297581	0.303854	0.329337	0.350374	0.321636	0.298385
CT	0.118235	0.393320	0.373500	0.119765	0.092748	0.433354	0.346589	0.085553
CT$_N$	0.431205	0.430389	0.457698	0.217162	0.197078	0.808499	0.786097	0.214751
SG	0.198457	0.201441	0.202078	0.195709	0.198113	0.210321	0.209603	0.197303
CSTR	0.013023	0.012234	0.012208	0.011749	0.014428	0.014618	0.014389	0.013708
CSTR$_N$	0.123870	0.112420	0.108313	0.123810	0.123126	0.107138	0.107278	0.119143
RA	0.107644	0.111001	0.110285	0.088419	0.129527	0.208327	0.205474	0.122716

VI. Conclusions, Discussion, and Further Work

We have introduced ensembles of fuzzy linear model trees for modeling multiple-output dynamic systems. In contrast to the abundant related work which models MIMO systems using separate MISO models for each output, this work proposes and evaluates the usage of multi-output Takagi-Sugeno models, which share a common structure for all outputs.

The multi-output Takagi-Sugeno model is built by an extension of the Lolimot model tree algorithm, which requires the setting of only one parameter and is capable of simultaneously modeling multiple outputs. The resulting multi-output model tree is converted to a multi-output Takagi-Sugeno model. We propose and evaluate two novel methods for creating a multi-output Takagi-Sugeno model: learning a multi-output model tree and learning an ensemble of multi-output model trees. We also evaluate the split selection heuristic of the Lolimot algorithm designed for dynamic systems and based on simulation error in comparison to the one-step ahead prediction error.

The evaluation is performed on six case studies of modeling multi-input multi-output nonlinear dynamic systems. One case study concerns the inverse dynamics of a robot arm, while the other five case studies deal with problems from process industries: a continuous-stirred tank reactor, a steam generator system, a gas-liquid separator, an industrial winding process, and four cascaded tanks. The models are built using noisy data, which presents a realistic modeling scenario.

The evaluation of the ensemble method performance in terms of simulation (output) error, shows that multi-output ensembles can successfully model the mentioned nonlinear dynamic systems. The ensembles consistently improve the performance over single tree models, both for single-output and multi-output models. The use of a selection heuristic based on simulation (output) error can bring slight improvement in output error performance in the best case. In the worst case, it shows performance similar to using a selection heuristic based on prediction error.

Decision trees tailored to system identification. The Lolimot algorithm uses an important feature that distinguishes it from the state-of-the-art decision tree learning methodology: a heuristic function that is tailored to system identification problems. A similar idea has been implemented and investigated in the context of decision trees that take spatial autocorrelation into account [58], where the problem of interest is the modeling of spatial data. Lolimot focuses on modeling temporal data.

The heuristic function for selecting splits in Lolimot uses the output error of the model, since the intended application of Lolimot is the modeling of dynamic systems. Our experimental evaluation shows that the selection heuristic influences the performance of both single trees and ensembles. Using the simulation heuristic yielded slightly improved performance, especially on the noisy CSTR$_N$ dataset.

Ensembles of Lolimot model trees. While ensembles of Takagi-Sugeno models [37] exist, ensembles of model trees for identification of Takagi-Sugeno models for multiple outputs have not been proposed or evaluated yet. In this work, we evaluate the performance of ensembles of multi-output Lolimot model trees for the construction of multi-output Takagi-Sugeno models. In particular, we apply bagging of multi-output Lolimot model trees. The results of the experimental comparison show that bagging in most cases increases the performance over a single tree, and this holds across all case studies, for both performance measures (output error and prediction error), and for both single- and multi-output trees.

Simultaneous predictions of multiple outputs. We compare the multi-output models to a set of r single-output models, each predicting one output: the predictive performance of a multi-output model is similar or slightly worse to that of a set of single-output models.

A multi-output model requires a smaller number of parameters for the structure of the Takagi-Sugeno models, as compared to a set of single-output models. In cases where the number of parameters for the structure is important, it is worth considering multi-output Takagi-Sugeno models, as their performance would not be much worse than that of the single-output alternative.

In sum, MO trees and ensembles of MO trees construct smaller models (with smaller number of parameters). However, in some cases the predictive performance of the multi-output alternative is slightly worse than the standard single-output variant. Having this in mind, we can overall recommend the use of bagging of SO trees learned by using the simulation error as a heuristic.

Further work. In further work, we would like to evaluate different ensemble approaches, such as boosting, using Lolimot trees. Also, we would like to explore the benefit of building single-output models in parallel: The results of the noisy CSTR case study show that it might be beneficial to build the single-output trees in parallel while using a selection heuristic based on simulation error. The simulation procedure, performed in each iteration, could use all partially built single-output models, instead of the measured data. This could improve the output error performance of a set of single-output models.

Also, a possibility for future research lies in exploring different non axis-parallel splitting strategies, as for example in SuHiClust, an approach previously discussed. In this context, the next perceived step in research is to evaluate ensembles using SuHiClust models.

Acknowledgments: This work was supported by the Slovenian Research Agency (grants P2-0103 and P2-0001) and the European Commission (grant ICT-2013-612944 MAESTRA). Thanks to Oliver Nelles for providing source code of Lolimot.

References

[1] K. J. Åström and B. Wittenmark, *Adaptive control*, 2nd ed. Reading, Mass: Addison-Wesley, 1995.

[2] L. Ljung, *System Identification: Theory for the User*. Prentice Hall, 1999.

[3] O. Nelles, *Nonlinear System Identification with Local Linear Neuro-Fuzzy Models*. Shaker, 1999.

[4] J. Abonyi, *Fuzzy model identification for control*. Birkhuser, 2003.

[5] F. Serdio, E. Lughofer, K. Pichler, T. Buchegger, M. Pichler, and H. Efendic, "Fault detection in multi-sensor networks based on multivariate time-series models and orthogonal transformations," *Information Fusion*, vol. 20, pp. 272 – 291, 2014.

[6] R. Murray-Smith and T. Johansen, Eds., *Multiple Model Approaches to Modelling and Control*. Taylor & Francis, 1997.

[7] T. Takagi and M. Sugeno, “Fuzzy identification of systems and its applications to modeling and control,” *IEEE Transactions on Systems, Man and Cybernetics*, no. 1, pp. 116–132, 1985.

[8] O. Nelles, *Nonlinear System Identification: from classical approaches to neural networks and fuzzy models*. Springer, 2001.

[9] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and regression trees*. CRC press, 1984.

[10] Y. Wang and I. H. Witten, “Inducing model trees for continuous classes,” in *Poster Papers of the 9th European Conference on Machine Learning*. Springer, 1997, pp. 128–137.

[11] E. Frank, Y. Wang, S. Inglis, G. Holmes, and I. H. Witten, “Using model trees for classification,” *Machine Learning*, vol. 32, no. 1, pp. 63–76, 1998.

[12] L. Torgo, “Functional models for regression tree leaves,” in *Proceedings of 14th the International Conference on Machine Learning*. Morgan Kaufmann, 1997, pp. 385–393.

[13] W.-Y. Loh, “Regression trees with unbiased variable selection and interaction detection,” *Statistica Sinica*, vol. 12, no. 2, pp. 361–386, 2002.

[14] A. Dobra and J. Gehrke, “Secret: a scalable linear regression tree algorithm,” in *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2002, pp. 481–487.

[15] J. Gama, “Functional trees,” *Machine Learning*, vol. 55, no. 3, pp. 219–250, 2004.

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

[17] E. Ikonomovska, J. Gama, and S. Džeroski, “Incremental multi-target model trees for data streams,” in *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, 2011, pp. 988–993.

[18] A. Appice and S. Džeroski, “Stepwise induction of multi-target model trees,” in *Proceedings of the 18th European Conference on Machine Learning*. Springer, 2007, pp. 502–509.

[19] J.-S. R. Jang, C.-T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing–A Computational Approach to Learning and Machine Intelligence*. Prentice Hall, 1997.

[20] F. Hoffmann and O. Nelles, “Genetic programming for model selection of TSK-fuzzy systems,” *Information Sciences*, vol. 136, no. 14, pp. 7 – 28, 2001.

[21] J. R. Quinlan, “Learning with continuous classes,” in *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, vol. 92. World Scientific, 1992, pp. 343–348.

[22] C. Marsala, “Data mining with ensembles of fuzzy decision trees,” in *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2009, pp. 348–354.

[23] A. Lemos, W. Caminhas, and F. Gomide, “Evolving fuzzy linear regression trees with feature selection,” in *Proceedings of the IEEE Workshop on Evolving and Adaptive Intelligent Systems*. IEEE, 2011, pp. 31–38.

[24] A. Suarez and J. Lutsko, “Globally optimal fuzzy decision trees for classification and regression,” *IEEE Transactions on Pattern Analysis and Machine Intelligence,*, vol. 21, no. 12, pp. 1297–1311, 1999.

[25] C. Olaru and L. Wehenkel, “A complete fuzzy decision tree technique,” *Fuzzy Sets and Systems*, vol. 138, no. 2, pp. 221–254, 2003.

[26] H. Blockeel and L. De Raedt, “Top-down induction of first-order logical decision trees,” *Artificial intelligence*, vol. 101, no. 1, pp. 285–297, 1998.

[27] A. Krogh and J. Vedelsby, “Neural network ensembles, cross validation, and active learning,” in *Proceedings of the 8th International Conference on Advances in Neural Information Processing Systems*. MIT Press, 1995, pp. 231–238.

[28] T. G. Dietterich, “Ensemble learning,” *The handbook of brain theory and neural networks*, pp. 405–408, 2002.

[29] J. Mendes-Moreira, C. Soares, A. M. Jorge, and J. F. D. Sousa, “Ensemble approaches for regression: A survey,” *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, p. 10, 2012.

[30] L. Breiman, “Bagging predictors,” *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[31] ——, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[32] B. Pfahringer, “Semi-random model tree ensembles: An effective and scalable regression method,” in *Australasian Conference on Artificial Intelligence*, 2011, pp. 231–240.

[33] M. Jung, M. Reichstein, and A. Bondeau, “Towards global empirical upscaling of fluxnet eddy covariance observations: validation of a model tree ensemble approach using a biosphere model,” *Biogeosciences*, vol. 6, no. 10, pp. 2001–2013, 2009.

[34] D. Aleksovski, J. Kocijan, and S. Džeroski, “Model tree ensembles for modeling dynamic systems,” in *Proceedings of the Sixteenth International Conference on Discovery Science (DS 2013)*, 2013, pp. 17–32.

[35] P. Bonissone, J. Cadenas, M. Garrido, and R. Dıaz-Valladares, “A fuzzy random forest: Fundamental for design and construction,” in *Proceedings of the 12th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU08)*, 2008, pp. 1231–1238.

[36] C. Z. Janikow and M. Faifer, “Fuzzy decision forest,” in *Proceedings of the 19th International Conference of the North American Fuzzy Information Processing Society (NAFIPS).* IEEE, 2000, pp. 218–221.

[37] M. Korytkowski, L. Rutkowski, and R. Scherer, “Rule base normalization in Takagi-Sugeno ensemble,” in *Proceedings of the 2011 IEEE Workshop On Hybrid Intelligent Models And Applications.* IEEE, 2011, pp. 1–5.

[38] F. Gasir, Z. Bandar, and K. Crockett, “An architecture for constructing fuzzy regression tree forests using opt-ainet,” in *Proceedings of the IEEE International Conference on Fuzzy Systems.* IEEE, 2011, pp. 283–289.

[39] R. Babuška, C. Fantuzzi, U. Kaymak, and H. Verbruggen, “Improved inference for Takagi-Sugeno models,” in *Proceedings of the Fifth IEEE International Conference on Fuzzy Systems*, vol. 1. IEEE, 1996, pp. 701–706.

[40] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, “Construction of fuzzy classification systems with rectangular fuzzy rules using genetic algorithms,” *Fuzzy sets and systems*, vol. 65, no. 2, pp. 237–253, 1994.

[41] D. Gustafson and W. Kessel, “Fuzzy clustering with a fuzzy covariance matrix,” in *IEEE CDC*, no. 17, 1978, pp. 761–766.

[42] B. Hartmann, O. Bänfer, O. Nelles, A. Sodja, L. Teslić, and I. Škrjanc, “Supervised hierarchical clustering in fuzzy model identification,” *Fuzzy Systems, IEEE Transactions on*, vol. 19, no. 6, pp. 1163–1176, 2011.

[43] T. A. Johansen and R. Babuska, “Multiobjective identification of Takagi-Sugeno fuzzy models,” *IEEE Transactions on Fuzzy Systems*, vol. 11, no. 6, pp. 847–860, 2003.

[44] E. Lughofer, *Evolving fuzzy systems-methodologies, advanced concepts and applications.* Springer, 2011.

[45] ——, “On-line assurance of interpretability criteria in evolving fuzzy systems - Achievements, new concepts and open issues,” *Information Sciences*, vol. 251, pp. 22–46, 2013.

[46] J. Demšar, “Statistical comparisons of classifiers over multiple data sets,” *The Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.

[47] M. J. van der Laan, S. E. Sinisi, and M. L. Petersen, “Cross-validated bagged learning,” Division of Biostatistics, University of California, Berkeley, Tech. Rep. 182, June 2005.

[48] J. Kocijan and B. Likar, “Gas–liquid separator modelling and simulation with Gaussian-process models,” *Simulation Modelling Practice and Theory*, vol. 16, no. 8, pp. 910–922, 2008.

[49] J. Kocijan and A. Grancharova, “Gaussian process modelling case study with multiple outputs,” *Comptes Rendus de l’Academie Bulgare des Sciences*, vol. 63, pp. 601–607, 2010.

[50] T. Bastogne, H. Garnier, and P. Sibille, “A pmf-based subspace method for continuous-time model identification. application to a multivariable winding process,” *International Journal of Control*, vol. 74, no. 2, pp. 118–132, 2001.

[51] B. De Moor, "Database for the identification of systems (DaISy)," Department of Electrical Engineering, ESAT/SCD, KU Leuven, Belgium, http://www.esat.kuleuven.ac.be/sista/daisy, 2010, Accessed September 2013.

[52] R. Babuska, J. Roubos, and H. Verbruggen, "Identification of MIMO systems by input-output TS fuzzy models," in *Proceedings of the 1998 IEEE International Conference on Fuzzy Systems*, vol. 1. IEEE, 1998, pp. 657–662.

[53] R. Babuska, "Fuzzy identification toolbox for MATLAB," Faculty of Mechanical Engineering and Systems, Delft University of Technology, The Netherlands http://www.dcsc.tudelft.nl/ rbabuska/, Accessed January 2015.

[54] J. J. Espinosa and J. Vandewalle, "Predictive control using fuzzy models," in *Advances in Soft Computing*. Springer, 1999, pp. 187–200.

[55] G. Pellegrinetti and J. Bentsman, "Nonlinear control oriented boiler modeling-a benchmark problem for controller design," *IEEE Transactions on Control Systems Technology*, vol. 4, no. 1, pp. 57–64, 1996.

[56] G. Lightbody and G. W. Irwin, "Nonlinear control structures based on embedded neural system models," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 553–567, 1997.

[57] S. Vijayakumar and S. Schaal, "Locally weighted projection regression: Incremental real time learning in high dimensional space," in *Proceedings of the Seventeenth International Conference on Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann, 2000, pp. 1079–1086.

[58] D. Stojanova, M. Ceci, A. Appice, D. Malerba, and S. Džeroski, "Dealing with spatial autocorrelation when learning predictive clustering trees," *Ecological Informatics*, vol. 13, no. 0, pp. 22 – 39, 2013.