Model Tree Ensembles for Modeling Dynamic Systems

Darko Aleksovski[1,3], Juš Kocijan[1,2], and Sašo Džeroski[1,3]

[1] Jožef Stefan Institute, Jamova 39, Ljubljana, Slovenia
[2] University of Nova Gorica, Vipavska 13, Nova Gorica, Slovenia
[3] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
{darko.aleksovski,jus.kocijan,saso.dzeroski}@ijs.si

Abstract. We address the task of discrete-time modeling of nonlinear dynamic systems using measured data. In the area of control engineering, this task is typically converted into a classical regression problem, which can then be solved with any nonlinear regression approach. As tree ensembles are a very successful predictive modelling approach, we investigate the use of tree ensembles for regression for this task.

While ensembles of regression trees have been extensively used and different variants thereof explored (such as bagging and random forests), ensembles of model trees have received much less attention, being limited mostly to bagging of model trees. We introduce a novel model tree ensemble approach to regression, namely, bagging and random forests of fuzzified model trees. The main advantage of the new approach is that it produces a model with no discontinuities with a satisfactory extrapolation behavior, needed for modeling dynamic systems.

We evaluate existing tree ensemble approaches to regression and the approach we propose on two synthetic and one real task of modeling nonlinear dynamic systems coming from the area of control engineering. The results show that our proposed model tree ensembles outperform ensembles of regression trees and have comparable performance to state-of-the-art methods for system identification typically used in control engineering. The computing time of our approach is comparable to that of the state-of-the-art methods on the small problems considered, with the potential to scale much better to large modeling problems.

1 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,

J. Fürnkranz, E. Hüllermeier, and T. Higuchi (Eds.): DS 2013, LNAI 8140, pp. 17–32, 2013.

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.

The Task of Discrete-Time Modeling of Dynamic Systems

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 [1], 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 $\mathbf{x}(k)$

$$\mathbf{x}(k) = [u(k-1), u(k-2), .., u(k-n), y(k-1), y(k-2), .., y(k-n)]^T \quad (1)$$

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., $\mathbf{x}(k)$). 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.

Early research in discrete-time system identification focused on parametric and semi-parametric regression approaches. More recently, non-parametric approaches dominate. These include the widely used basis-function approaches of Artificial Neural Networks [1] and fuzzy modeling, as well as kernel methods [2] and Gaussian Process models [3].

In this context, we propose the use of another non-parametric approach to regression, coming from the area of machine learning. Tree ensembles are very well known and perform well in terms of predictive power. They are also very efficient and scale well to large problems. We thus set out to investigate the use of tree ensembles for regression for the above task.

To our knowledge, the present work is the first to consider the use of tree ensemble methods for modeling dynamic systems.

Tree Ensembles for Regression

The simplest ensemble creation procedures is based on the resampling principle and is known as Bootstrap Aggregation (Bagging) [4]. It generates ensembles from randomly sampled bootstrap replicates (random samples with replacements) of the training set. Each bootstrap replicate is used to build a base model, using the base learner algorithm. Hopefully, each base model would capture and represent a different hypothesis and the combination of the base models would be a more accurate predictor than each base model. Bagging can be seen as an ensemble method that modifies the training data.

Later work by Breiman [5] concluded that ensembles could benefit from modifying the learning process as well. This tree ensemble methodology is known as Random Forests and includes randomization of the attributes considered for split nodes of the trees. Both tree ensemble approaches use a regression tree learning algorithm as a base learner, which only provides a piecewise constant approximation of the true function.

However, the modeling of nonlinear dynamic systems using the transformation to a regression task requires approximating (smooth) nonlinear functions. Also, the extrapolation behavior of the model is important because the measured (training) data for the dynamic system cannot capture all parts of the operating region being modeled [6]. The existing piecewise constant fit that a regression tree or an ensemble of regression trees provide is not sufficient for this task. An obvious solution of extending the ensembles of regression trees methodology is introducing ensembles of model trees, which learn linear models for the terminal nodes.

Linear model trees provide a piecewise linear approximation and are a more powerful approximator which also handles the extrapolation problem better than regression trees. Another approach that is often pursued in the system identification community, for modeling dynamic systems, is utilizing fuzzy methods. In the context of model trees, fuzzy approaches are able to provide continuous transitions between neighboring local (linear) models and a more accurate fit to the true nonlinear function.

While ensembles of regression trees have been used extensively and different variants thereof have been explored [7,8,5], ensembles of model trees have received much less attention in the machine learning literature. In fact, they have been mostly limited to bagging of model trees. Here we introduce a novel model tree ensemble approach to regression, namely, bagging and random forests of fuzzified model trees, which produce models with no discontinuities and with satisfactory extrapolation behavior.

Paper Outline

In this paper, we introduce a novel model tree ensemble approach to regression, namely, (bagging and) random forests of fuzzified model trees (Section 2). The main advantage of the new approach is that it produces a model with no discontinuities with a satisfactory extrapolation behavior. As explained above, this is needed for modeling dynamic systems.

We then evaluate existing tree ensemble approaches to regression and the approach we propose on two synthetic and one real task of modeling nonlinear dynamic systems coming from the area of control engineering. We also consider two state-of-the-art approaches to system identification coming from that area. Section 3 describes the experimental setup, while Section 4 presents the results.

The results show that our proposed model tree ensembles outperform ensembles of regression trees. They have comparable performance to the two state-of-the-art methods for system identification that are typically used in control engineering. The computing time of our approach is comparable to that of the state-of-the-art methods on the small problems considered, with the potential to scale much better to large modeling problems.

2 Model Trees Ensembles

The Model Trees Ensembles (MTE) methodology works by learning fuzzified linear model trees from bootstrap samples of the training data. The pseudocode for the Model Tree Ensembles methodology is shown in Table 1. The base learner algorithm starts by learning crisp linear model trees, where an example follows exactly one branch of a split in the tree for tests of the form $[A < v]$ (v is a crisp threshold). The next stage involves a tree fuzzification procedure, which transforms each (crisp) split of the trees into a fuzzy split, where an example can be sorted down both branches resulting from a split and the two corresponding predictions are combined. The prediction of the model tree ensemble $E = \{T_1, T_2, .., T_m\}$ is a uniformly weighted average of the individual tree predictions:

$$\hat{f}(\boldsymbol{x}) = \frac{1}{m} \sum_{T \in \{T_1, T_2, .., T_m\}} \hat{f_T}(\boldsymbol{x}) \tag{2}$$

where $\hat{f_T}$ is the prediction of model tree T.

The crisp model tree learning is based on the well-known M5′ algorithm [9,10]. The randomized base learner for building fuzzified model trees operates in the following stages: tree growing phase, tree pruning phase, and tree fuzzification phase. The tree growing phase includes a randomization of the split attribute selection, in line with the ideas from Random Forests [5]. The different stages of the tree learning algorithm are described in the sections that follow, and the pseudocode is shown in Tables 2, 3 and 4.

2.1 Tree Growing Phase

Tree growing 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 (t). 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 (3) below. Normally the split with the highest reduction in the standard deviation is chosen. However, a randomization of the split attribute selection process can also be implemented in the tree growing phase, as described below.

Randomization of Attribute Selection. We consider a randomization of the base learning algorithm as in the random forests approach: instead of considering all features, only a random subset of features is considered when selecting the best split in a given node. A different random subset of the same size is considered in different nodes. The size is determined by the attribute randomization parameter p. For each candidate attribute in the subset, 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:

$$SDR = \sigma_S^2 - \frac{|S_l|}{|S|}\sigma_{S_l}^2 - \frac{|S_r|}{|S|}\sigma_{S_r}^2 \tag{3}$$

where S is the set of data points falling in the current tree node, S_l and S_r are the two subsets of data points corresponding to the left and right branches of the split. σ_S^2 denotes the standard deviation of the target attribute in the set S.

2.2 Tree Pruning Phase

It is well known that overly large trees are prone to overfitting. Tree pruning (shown in Table 3) is a method that handles overfitting by removing tree nodes which may deteriorate the performance of the tree. As a first step, linear models are estimated in all nodes of the tree, by using linear regression. The estimation of linear models also includes an attribute removal part: Features whose effect is small are dropped from the linear model [9].

After the estimation of linear models, the bottom-up pruning function evaluates whether to prune each tree node. It compares the accuracy of the linear model learnt for a 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.

Our experience showed that including unpruned model trees in the ensemble resulted in deterioraion of the ensemble performance. The linear regression procedure for the terminal nodes that contained smaller number of data points resulted in invalid linear models. One possible solution to this problem

was to include a tree pruning method, which increased the performance of the ensembles.

2.3 Fuzzification

In order to smooth out the discontinuous transitions between two neighboring linear models of a model tree (i.e. linear models placed in two nodes following the same split node), we implement a split fuzzification procedure. A crisp split of the form $s[A < c]$, where A is a descriptive attribute and c is a cut-point, is transformed to a fuzzy split: A data instance now belongs to both the left and right subtrees of the split with probabilities $\mu_{A<c}$ and $\mu_{A\geq c}$ respectively. The probabilities for membership to the left and right subtrees are calculated by using a sigmoidal membership function with a single parameter: α - the inverse split width, as defined by (4) below.

$$\mu_{A<c} = \frac{1}{1+\exp(-\alpha(A-c))}, \qquad \mu_{A\geq c} = 1 - \mu_{A<c} \tag{4}$$

The value of the split parameter α is different for each split. It is calculated so that the overlap between the two neighboring partitions that the split creates in the dimension of the split variable A is equal to some percentage w of the range of values for that attribute. In the experiments, the value of the parameter w is optimized using cross-validation.

2.4 Implementation

The Model Tree Ensembles method is implemented in Java, using the WEKA data mining software [11]. The WEKA implementation of the M5′ algorithm [9] for building model trees has been modified to include the changes described here.

In the Model Tree Ensembles method we extend the M5′ algorithm, but with some of its features disabled. The disabled features consider the learning of the linear models and the smoothing procedure.

The linear models in a node of an M5′ tree are learnt only using variables found in tests in the subtree rooted at that node. Instead of keeping this feature we learn the linear models by using all available features. Also, we turn off the smoothing procedure of M5′. Our preliminary experiments have shown that there is no difference in performance between ensembles which include and those that do not include the two features of the base learner.

3 Experimental Setup

In this section, we describe the datasets used for the experimental evaluation, the methods that we compare, the optimization of the parameters and the metrics used for the evaluation of the results.

Table 1. Pseudocode for the top level of the Model Tree Ensembles method

Learn_ensemble(S)
Input: S - training set
Output: E - an ensemble
Create m bootstrap samples of $S : S_1, S_2, .., S_m$
For $k = 1 \rightarrow m$ *do*
    $T_k =$**Build_tree**(S_k)
    $T_k =$ **Prune**(T_k)
Let $w_{opt} =$ **Optimize_overlap_width_parameter**($\{T_1, T_2, .., T_m\}, S$)
For $k = 1 \rightarrow m$ *do*
    $T_k =$ **Fuzzify**(T_k, w_{opt})
Return ensemble $E = \{T_1, T_2, .., T_m\}$

3.1 Datasets

We have used three dynamic system datasets from the area of control engineering, more precisely process-industries. One of the datasets is measured, while the other two are synthetic. Of the latter, one is noise-free and the other is noisy.

The first dataset concerns the modeling of a unit for separating gas from liquid, which is part of a larger pilot plant [12]. It is a dataset with measured values concerning a semi-industrial process plant. The dynamic system is represented by 4 variables: three input (state of first valve, state of second valve, level of liquid in tank) and one system (pressure of gas in tank) variable. The dynamic order (lag) selected for this dataset is 1, which means that the regression dataset contains 4 features and 1 target variable. The dataset is already split into a training and test part, each of which contains 733 data instances.

The second dataset is generated from a dynamic system model that concerns the control of pH neutralization [13]. This synthetic dataset contains one input variable and one system variable. The selected dynamic order (lag) is 2, which means that the regression dataset contains 4 features and 1 target variable. This dataset too is split into a training and a test part, each having 320 data points. The training and testing sets for both datasets are obtained from different signals generated under the same conditions.

The third dataset is generated from the same dynamic system model as the second one, but with added noise. In more detail, the noise is added only to the system variable and only in the training set. In the final regression dataset, this means that two of the four feature variables (the ones corresponding to the system variable of the dynamic system model), as well as the target variable, contain noise in the training set. The noise added is white noise with a standard deviation equal to 20% of the standard deviation of the target variable.

3.2 Methods and Parameter Settings

We compare MTEs with two selected methods typically used in system identification. The first method that we compare to is the method of Neural Networks

Table 2. Pseudocode for the tree growing phase of the Model Tree Ensembles method

Build_tree(S)
Input: S - a training set
Output: T - a tree
If $|S| < t$
    Return a terminal node
If standard deviation pre-pruning criterion is satisfied
    Return a terminal node
Let $\{A_1, A_2, .., A_p\}$ be a random subset of feature attributes
Initialize s_{best}
For $k = 1 \rightarrow p$ *do*
    Let split $s^*[A_k < c] = \underset{s}{argmax}(SDR(s[A_k < c]))$
    If $SDR(s^*) < SDR(s_{best})$
        $s_{best} = s^*$
Split set S into subsets S_l and S_r based on split s_{best}
Let $T_l =$ **Build_tree**(S_l), $T_r =$ **Build_tree**(S_r)
Return a tree with a split node s_{best} and subtrees T_l and T_r

Table 3. Pseudocode for the tree pruning phase of the Model Tree Ensembles method

Prune(T)
Input: T - a tree
Output: pruned tree
If root of T is a split node
    Prune($T \rightarrow$ left)
    Prune($T \rightarrow$ right)
    Learn a linear model for the root node of T
    Calculate the error of the linear model err_{LM}
    If **Subtree_error**(T) $> err_{LM}$ then
        Convert root of T to a terminal node
Return T

Subtree_error(T)
Input: T - a tree
Output: numeric value
If root of T is a split node
    Let T_l=$T \rightarrow$ left, T_r=$T \rightarrow$ right
    Let S=$T \rightarrow$ examples, S_l=$T_l \rightarrow$ examples, S_r=$T_r \rightarrow$ examples
    Return ($|S_l|$ * **Subtree_error**(T_l) + $|S_r|$ * **Subtree_error**(T_r)) / $|S|$
Otherwise
    Return the error of the linear model in root of T

Table 4. Pseudocode for the fuzzification phase of the Model Tree Ensembles method

Fuzzify(T, w)
Input: T - a tree, and w - overlap width parameter
Output: fuzzified tree
If root of T is a split node
    Let the split at root of T be $s[A_k < c]$
    Let $[x_k^{min}, x_k^{max}]$ be the range of the data points in dimension k
    Calculate α s.t. split width is equal to $w|x_k^{max} - x_k^{min}|$
    Create fuzzy split at root of T with parameters α, A_k, c using Eq.4
    Fuzzify$(T \rightarrow$ left$)$
    Fuzzify$(T \rightarrow$ right$)$
Return T

Optimize_overlap_width_parameter(E, S)
Input: E - an ensemble, and S - a training set
Output: optimal overlap width
Let $w = [10\%, 20\%, 30\%, .., 90\%]^T$
For $k = 1 \rightarrow 9$ do
    For each tree T in ensemble E do
      $T =$ Fuzzify(T, w_k)
    Let err_k be a cross-validation estimate of the error of ensemble E, using set S
Let $p = \underset{k}{argmin}(err_k)$
Return w_p

(NN). We use feed-forward multi-layer perceptron (MLP) networks [1] with one hidden layer of nodes. The only parameter that we tune for NN is the number of nodes in the single hidden layer. We test values for this parameter in the range from 1 to 10, so in total we test 10 possible values. For this work we don't anticipate a need for more than 10 hidden neurons or the need for more than one hidden layer. The implementation of the multi-layer perceptron networks that we use is the one from the Neural Network Toolbox in Matlab.

The second method is the Adaptive Network Based Fuzzy Inference System (ANFIS) [14]. It is a hybrid neural-network approach, which combines backpropagation gradient descent and least-squares regression. The model it builds is a set of fuzzy rules (Takagi-Sugeno fuzzy model). The premise parts of the rules are built using fuzzy sets, while the consequents are linear models of the feature attributes.

The method first determines the number of fuzzy rules and their initial positions (structure identification part). This is done by using a fuzzy c-means clustering algorithm. After the initial stage, the method applies an iterative improvement of the model, consisting of two steps. The first step is backpropagation gradient descent, which modifies the parameters of the fuzzy membership functions (it re-locates the centers of the fuzzy rules and changes the parameters of the Gaussian membership functions). The second step is learning the parameters of the linear models of the rule consequents. For this step a weighted

least-squares regression is applied, and the linear model parameters of all rules are learnt simultaneously.

The only parameter that needs to be set for ANFIS is the number of rules (clusters). The Matlab implementation (available in the Fuzzy Logic Toolbox) we use has the option to determine automatically the number of rules, but in our experience suboptimal results are obtained in that way. We thus tune the value of this parameter (which is in the range from 2 to 10) by considering 9 possible integer values for it.

The parameters that need tuning for MTE are the following a) the minimal number of examples t, b) the attribute randomization parameter p and c) the overlap width parameter w. First, we perform a search for the optimal combination of the two parameters (t_{opt}, p_{opt}) using internal cross-validation of ensembles of 10 trees. Once this pair has been found, we optimize the overlap width parameter w. We consider 9 different values for this parameter, ranging from 10% to 90%. After the parameter optimization step, the training of the ensembles is carried out, by learning 50 trees. The total number of parameter combinations tried for each of the datasets is reported in Section 4.2.

We also consider four alternatives of our method: bagging of model trees (BMT), bagging of fuzzified model trees (BTM+fz), (random) forests of model trees (FMT) and (random) forests of fuzzified model trees (FMT). In addition, we consider several baselines closely related to our MTE method. The baseline methods chosen are: a single M5′ model tree (M5′), bagging of M5′ regression trees (BRT) and (random) forests of M5′ regression trees (FRT). The parameters of the baseline methods are optimized using the same internal cross-validation procedure as for the MTE method. The baseline methods we consider are implemented in WEKA [11].

3.3 Evaluation Criterion and Methodology

We evaluate the predictive performance and training time of the MTE method and compare these to the corresponding measures for NN and ANFIS. For the predictive performance, we calculate the squared prediction error, and report the root-relative mean-squared error (RRMSE), as defined by (5).

$$RRMSE = \frac{\sqrt{\sum (y_i - \hat{f}(x_i))^2}}{\sqrt{\sum (y_i - \bar{y})^2}} \tag{5}$$

Note that RRMSE normalizes the root mean squared error (RMSE) by the error of the single model that always predicts the average.

For the training time, we evaluate the time needed for parameter optimization and training (i.e. learning a regression model). The times are reported in seconds and all of the experiments are run on a machine with an Intel Pentium 4 CPU running at 3.2Ghz and 2GB of RAM memory.

The parameter optimization part of the learning is carried out by using 5-fold cross-validation. The prediction error of the cross-validation procedure is a criterion for selecting the best parameter or parameter combination. In order

to get a more reliable estimate of the performance of the methods with the selected parameters, we run the experiments 20 times using different random seeds. The performance values reported in Section 4 are the mean and standard deviations of the 20 experiment runs. Note that ANFIS uses randomization for setting the initial parameters of the fuzzy rules, NN use randomization to set the initial parameters of the network and MTEs use randomization for creating the bootstrap replicates and for split attribute randomization.

4 Results and Discussion

Here we report the experimental results, i.e. the predictive performance of the methods, the time for training needed, and the effect of the size of the ensembles (number of trees) on the predictive performance.

4.1 Predictive Performance

The predictive performance of the MTE method, its variants and the baseline methods we compare it to on the dynamic system datasets is shown in Table 5 and Figure 1. First we note that ensembles of regression trees perform much worse than all model-tree based methods. They perform worse than even a single model tree. This is in line with the perceived idea of smoothness and extrapolation for modeling dynamic systems. Note that bagging of model trees performs worse than an individual model tree, both for crisp and fuzzified trees. The only exception is ensembles of the pH20 datasets, where bagging of fuzzified trees performs better than a single model tree. Random forests of model trees (both crisp and fuzzy) perform better than bagging.

In fact, random forests of fuzzified model trees perform best for all three datasets. For the GLS and pH datasets, the improvement in predictive performance over that of a single tree is small. On the noisy dataset pH20, the improvement in performance is clearly visible. The fuzzification of the trees in the ensemble helps in decreasing the error, both for random forests and for bagging of model trees.

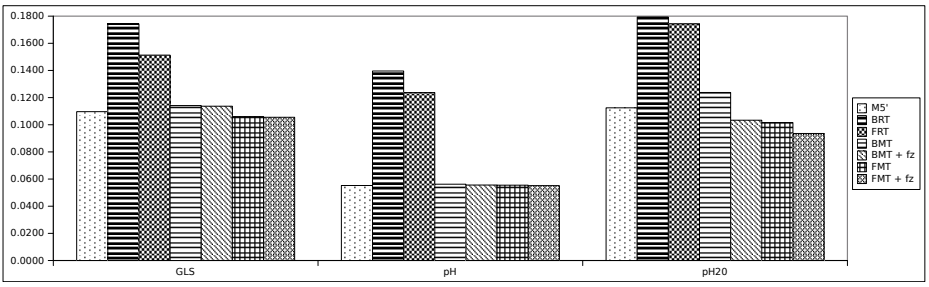

Fig. 1. Predictive performance of Model Tree Ensemble variants and baseline tree-based models

Table 5. Predictive performance of Model Tree Ensemble variants and baseline tree-based models

	GLS		pH		pH20	
	mean	st.dev.	mean	st.dev.	mean	st.dev.
M5′	0.1096	0.0000	0.0553	0.0000	0.1125	0.0000
BRT	0.1743	0.0017	0.1396	0.0030	0.1794	0.0048
FRT	0.1513	0.0036	0.1237	0.0091	0.1743	0.0049
BMT	0.1142	0.0056	0.0562	0.0021	0.1238	0.0027
BMT + fz	0.1137	0.0054	0.0556	0.0016	0.1033	0.0018
FMT	0.1060	0.0019	0.0555	0.0020	0.1016	0.0029
FMT + fz	**0.1055**	0.0019	**0.0551**	0.0017	**0.0936**	0.0018

The predictive performance of the MTE methodology is also compared to the selected methods, used in system identification, and described in Section 3.2. The results of the comparison are shown in Table 6 and Figure 2.

Table 6. Predictive performance of MTEs as compared to selected methods used in system identification

	GLS		pH		pH20	
	mean	st.dev.	mean	st.dev.	mean	st.dev.
FMT	0.1060	0.0019	0.0555	0.0020	0.1016	0.0029
FMT + fz	0.1055	0.0019	**0.0551**	0.0017	**0.0936**	0.0018
ANFIS	**0.1015**	0.0000	0.0751	0.0000	0.0937	0.0000
NN	0.1243	0.0350	0.0803	0.0095	0.1284	0.0139

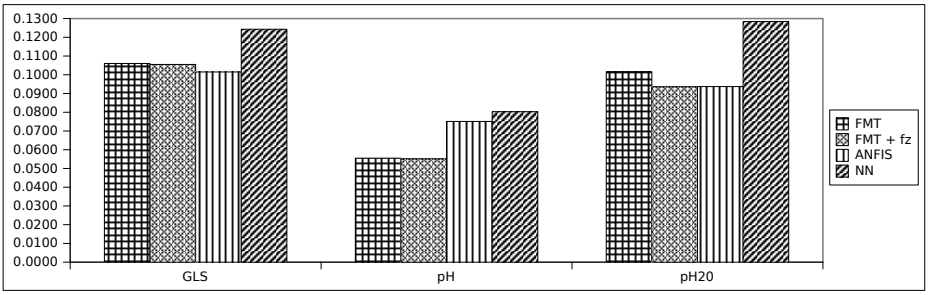

Fig. 2. Predictive performance of MTEs as compared to selected methods used in system identification

The comparison of the RRMSE for MTEs, ANFIS and NN shows that the performance of MTEs is better or comparable to ANFIS. On the GLS dataset, ANFIS is the winning method, but MTEs are the winning method in the other two cases. The performance of ANFIS is worse on the pH dataset, as the optimal number of fuzzy rules, as determined by the cross-validation procedure, is only 2. For comparison, the optimal number of fuzzy rules for the noisy counterpart, i.e. pH20 dataset, is 5, which leads to more competitive performance.

For the noisy pH20 dataset, MTEs show good performance, comparable to that of ANFIS. The standard deviation of the estimates of RRMSE error is almost zero for ANFIS and quite small for the MTE method. On the other hand, Neural Networks show quite a large deviation of the RRMSE result, because the random initial values used for the optimization of the network parameters lead to solutions with very different quality.

The optimal sets of parameters for the results presented in Tables 5 and 6 are as follows: For the FMT+fz variant of the MTEs, the value of the parameter t, the minimal number of examples, was 125, 50 and 90 for the three datasets, i.e. GLS, pH and pH20, respectively. The value of the attribute randomization parameter p was 1, 2 and 2, while the optimal overlap w had values of 20%, 20% and 30%. The number of trees included in all of the ensembles was 50. The optimal number of rules for ANFIS was 2, 2 and 5, while the optimal number of nodes in the hidden layer of NN was 6, 2 and 2 for the three datasets, respectively.

4.2 Computing Times

In this section, the time for parameter optimization and training is evaluated both separately and together for the three methods: MTE, ANFIS and NN. Table 7 summarizes the number of parameters that need tuning for each method and the number of possible combinations of parameter values that were tested in the 5-fold cross-validation procedure. It also reports the time needed for parameter optimization and for training (separately and together).

As discussed in Section 3.2 the MTE methodology has three parameters to tune and the total number of parameter value combinations tested ranges from 72 to 120. However, the tree building procedure is relatively fast, so the total time for model selection comparable to the time taken by the two selected methods. The results show that, in spite of the large number of parameter value combinations tested for MTEs, it takes less time to determine their optimal values and train the resulting model as compared to ANFIS. It can be noticed that the duration of the training procedure is shortest for NN, while it is longest for ANFIS. This is also visible in Figure 3.

Table 7. Computing times taken for parameter optimization and model training. The number of parameters tuned and the number of combinations of their values are listed as well.

		Parameter optimization			Training	Total
		time (sec)	num.param.	num.values	time (sec)	time (sec)
GLS	FMT+fz	230.0	3	120	2.6	232.6
	ANFIS	319.0	1	9	17.5	336.5
	NN	174.6	1	10	4.1	178.7
pH	FMT+fz	72.0	3	72	1.1	73.1
	ANFIS	197.5	1	9	7.5	205.0
	NN	45.9	1	10	1.0	46.9
pH20	FMT+fz	79.1	3	72	0.8	79.9
	ANFIS	198.1	1	9	7.7	205.8
	NN	33.8	1	10	0.6	34.4

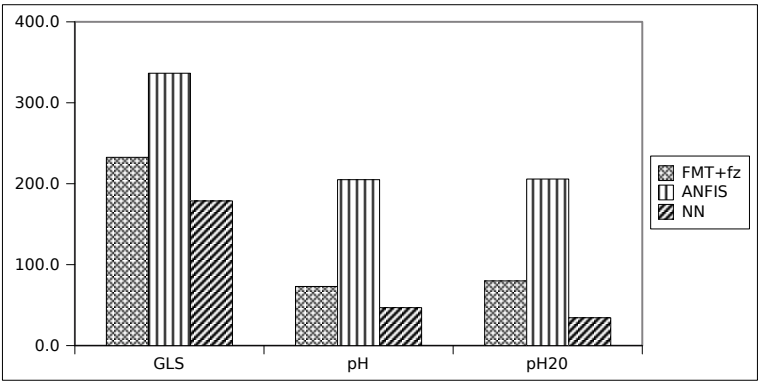

Fig. 3. Computing times

Note that decision trees scale very well as the number of training instances and the number of features increases. The same holds for ensembles of trees, especially for random forests. We thus expect our MTE methodology to perform more efficiently than both ANFIS and NN for larger problems.

4.3 The Effect of the Number of Trees in Ensemble

Table 8 and Figure 4 present the predictive performance (in terms of RRMSE) of MTEs with different number of trees. The results, obtained by using the FMT variant, show that the mean performance (out of 20 runs) increases slightly but not substantially by using more trees. This means that the optimal predictive performance can be obtained by using as few as 10 or 50 trees. In Table 8 however, it can be seen that the standard deviation of the error decreases when more trees are included in the ensemble.

Table 8. Effect of the number of trees in the ensemble on its performance

Num.trees in ensemble	GLS		pH		pH20	
	mean	st.dev.	mean	st.dev.	mean	st.dev.
10	0.1091	0.0044	0.0566	0.0032	0.1038	0.0059
50	0.1060	0.0019	0.0555	0.0020	0.1016	0.0029
100	0.1053	0.0020	0.0558	0.0011	0.1012	0.0025
500	0.1054	0.0007	0.0553	0.0006	0.1002	0.0010

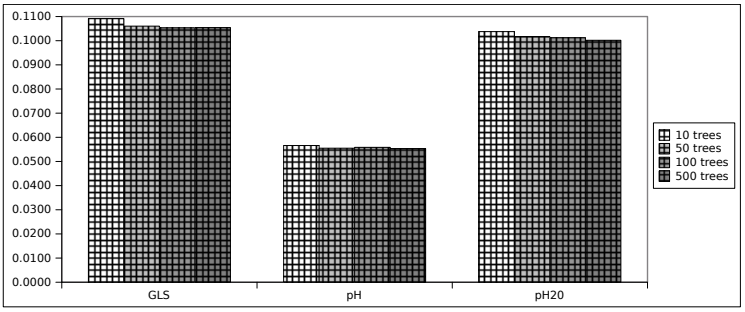

Fig. 4. Effect of the number of trees in the ensemble on its performance

5 Conclusions

In this paper, we investigated the task of modeling dynamic systems using tree ensembles. As it requires a modeling procedure which produces a smooth fit and a reasonable extrapolation behavior, we experimentally confirmed our hypothesis that regression trees and ensembles thereof are not appropriate. Instead, we introduced and evaluated ensembles of model trees with fuzzy splits.

Our experimental evaluation showed that the fuzzification procedure improves the performance of the model tree ensembles, especially on noisy data. Overall, the accuracy of the model tree ensembles was better than ensembles of regression trees and it also was better than a single model tree. Compared to selected state-of-the-art methods used in the area of system identification, the model tree ensembles performed equally well or better.

In our future work, we would like to evaluate the performance of ensemble models using a more stringent evaluation methodology tailored for modeling dynamic systems. This evaluation concerns the output error which is calculated by a procedure of simulation: the model at hand is repeatedly used for prediction in consecutive time points, with the feature vector at each time point modified to contain previous model predictions instead of the measured lagged values of the system variable. The evaluation in terms of output error would show whether the model predictions can be used for practical purposes or overfit noise and cause the simulation to diverge.

Acknowledgements. We would like to gratefully acknowledge the financial support of the following institutions: The Slovene Human Resources Development and Scholarship Fund, the Slovenian Research Agency (Grants P2-0001 and P2-0103), the European Commission (Grants ICT-2010-266722 and ICT-2011-287713), and the Operation no. OP13.1.1.2.02.0005 financed by the European Regional Development Fund (85%) and the Ministry of Education, Science, and Sport of Slovenia (15%).

References

1. Nelles, O.: Nonlinear system identification: from classical approaches to neural networks and fuzzy models. Springer (2001)
2. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press (2000)
3. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press (2006)
4. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
5. Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001)
6. Murray-Smith, R., Johansen, T. (eds.): Multiple Model Approaches to Modelling and Control. Taylor and Francis systems and control book series. Taylor and Francis, London (1997)
7. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Machine Learning 63(1), 3–42 (2006)
8. Breiman, L.: Randomizing outputs to increase prediction accuracy. Machine Learning 40(3), 229–242 (2000)
9. Wang, Y., Witten, I.H.: Inducing model trees for continuous classes. Poster Papers of the 9th European Conference on Machine Learning (ECML 1997), Prague, Czech Republic, pp. 128–137 (1997)
10. Quinlan, J.R.: Learning with continuous classes. In: Proceedings of the 5th Australian Joint Conference on Artificial Intelligence, vol. 92, pp. 343–348 (1992)
11. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. ACM SIGKDD Explorations 11(1), 10–18 (2009)
12. Kocijan, J., Likar, B.: Gas–liquid separator modelling and simulation with Gaussian-process models. Simulation Modelling Practice and Theory 16(8), 910–922 (2008)
13. Henson, M.A., Seborg, D.E.: Adaptive nonlinear control of a pH neutralization process. IEEE Transactions on Control Systems Technology 2(3), 169–182 (1994)
14. Jang, J.S.R., Sun, C.T., Mizutani, E.: Neuro-Fuzzy and Soft Computing–A Computational Approach to Learning and Machine Intelligence. Prentice Hall (1997)