Short-term Time Series Forecasting with Regression Automata

Qin Lin Christian Hammerschmidt[1] Gaetano Pellegrino Sicco Verwer

Department of Intelligent Systems
Delft University of Technology
Delft, the Netherlands
q.lin, g.pellegrino, s.e.verwer@tudelft.nl

Interdisciplinary Centre for Security, Reliability and Trust[1]
University of Luxembourg
Luxembourg
christian.hammerschmidt@uni.lu

ABSTRACT

We present regression automata (RA), which are novel type syntactic models for time series forecasting. Building on top of conventional state-merging algorithms for identifying automata, RA use numeric data in addition to symbolic values and make predictions based on this data in a regression fashion. We apply our model to the problem of hourly wind speed and wind power forecasting. Our results show that RA outperform other state-of-the-art approaches for predicting both wind speed and power generation. In both cases, short-term predictions are used for resource allocation and infrastructure load balancing. For those critical tasks, the ability to inspect and interpret the generative model RA provide is an additional benefit.

Keywords

Time Series Forecasting; Syntactic model; Regression Automata; (State) Machine Learning

1. INTRODUCTION

Forecasting is one of the most significant challenges in time series analysis [1]. Financial and wind power series attract continual attention, and many techniques were proposed and studied in the recent decades [2]. In this paper, we propose a novel model for learning syntactic patterns and forecasting such series. We apply our algorithm to wind speed and wind energy prediction problems.

With the promotion of sustainable energy production in many countries, wind energy generation is developing rapidly [3, 4]. Due to the wind's uncertainty and discontinuity, accurate wind speed prediction is one of the key challenges for safe and reliable operation of wind power systems [5, 6]. Typically, short-term wind prediction, on time spans less than 6 hours, and long-term wind prediction are distinguished. In this paper, we focus on the former, which is an important problem for energy storage system design, power dispatching, electricity pricing, etc.

KDD '16 August 13–17, 2016, San Francisco, California, USA

DOI: 10.1145/1235

During the past 30 years, many methods for wind speed prediction have been proposed. Generally those techniques can be classified into three categories. The first one is the conventional statistical model. Autoregressive Moving Average (ARMA) is the most representative [7]. Another conventional model is Kalman filter algorithm [8]. The second one is spatial correlation model. The original idea is called correlated echelon model (CEM) [9]. It proposed to combine time series forecasting and terrain characteristics. The techniques of third category are from the area of artificial intelligence and machine learning. The typical models that have been successfully applied in time series forecasting are neural networks [10], support vector machines [11], and fuzzy logic models [12] to name a few.

Syntactic models are alternatives to the conventional systems, because the learned models allow to inspect, interpret, and understand complex system dynamics [13]. Examples of such models are hidden Markov models (HMMs) and finite automata (FA) [14]. Syntactic methods are based on symbols that have typically been abstracted from numeric data in a pre-processing step. This gives three main advantages: firstly, categorical prediction reduces the computation cost. Secondly, raw time series data in practice tends to be very noisy. Symbolic representations are more robust to noise. Lastly, the category bounds can be modified to reflect prediction uncertainty, which is now becoming a trend in regression. To our best of knowledge, the only syntactic models applied in wind speed prediction are Markov chains [15] and semi-Markovian variants [16]. An interesting indirect approach to syntactic modeling of daily foreign exchange rates was proposed by Lee Giles et al [17]. They first abstracted the raw financial data into symbols using a SOM (self-organizing map), and then applied RNNs (recurrent neural networks) to the sequences for training. Finally, DFA (deterministic finite state automata) were extracted from RNNs for model interpretation. Unfortunately, this novel model was only able to be used to predict directionality, i.e., whether the exchange rate is positive or negative in the future. Another related work is SAX (Symbolic Aggregate approXimation), which provided high level representation for time series data [18]. However the main goals of SAX were dimensionality reductions and similarity measurements rather than forecasting.

Syntactic models are very useful because they provide a concise overview of numeric time series' behavior. A problem, however, is that they predict symbols instead of numeric values. Consequently, both their learning and prediction processes are less exact than those used by numeric

models and therefore more difficult to evaluate and harder to use in practice. In this paper, we overcome this problem of syntactic models by incorporating the numeric data values in the learning and prediction processes. Intuitively, the inputs of our model are the tuples of real numerical values and symbolical values abstracted from the raw data. The symbols are used for building the syntactic models underlying a time series' behavior in high level with states transitions, while the numeric values are used to accurately reflect the evolution of time series.

We preprocess the raw time series data sequentially and discretize the numeric values into abstract symbols. We then learn an RA using the DFASAT algorithm [19], but with a novel heuristic and a novel consistency criteria. Finally, we compare the resulting numeric predictions with baseline methods such as persistence, autoregressive integrated moving average (ARIMA), neural networks, and regression trees. The results demonstrate that our new method is competitive with these commonly used methods. Furthermore, they show that the numeric and syntactic prefix tree model used as input for DFASAT is already competitive with the state-of-the-art, albeit worse than the model obtained after learning. This result demonstrates the power of our method used to combine numeric and syntactic data for time series prediction.

Our contributions are the following:

- We develop a new method for learning DFA from time series data using both numeric and symbolic inputs. To our best knowledge, this is the first work that proposes to learn automata for numeric regression tasks.
- We propose a novel heuristic and consistency test for guiding the automaton learning process.
- We show that the learned models make predictions in real unseen data with high accuracy, outperforming the competition in short-term wind speed and wind power prediction.

This paper is organized as follows. Section 2 introduces data preprocessing. Section 3 briefly discusses the model building and learning algorithm. The experimental results are presented in Section 4. Section 5 discusses the results and concludes the paper.

2. DATA PREPROCESSING

2.1 Discretization

The numeric signal needs to be abstracted as symbols for state machine (automaton) learning. In this paper, we use SAX to descritize numeric data. Figure 1 illustrates a example of SAX. It firstly normalizes the raw data, then compresses by aggregating into piecewise aggregate approximations (PAAs). Lastly PAAs are assigned to symbols with quantiles of standard normal distribution. In this example, the raw data has length 48, the PAAs, i.e. colored bars have the same size of 12. We will finally get a frame with 4 letters "ccac". If we SAX the whole training data set in the beginning and then slice them into frames, we will call this strategy as "global SAX" in this paper. Table 1 shows the symbols and their corresponding numeric guards in the experimental case study one (see Section 4.3). All numeric values are abstracted to the symbol according to the bins they fall in. Note that we transform the bins of quantiles from standard normal distribution to un-normalized value for better explanation. We use the similar idea of "ELBOW" method [20] to determine the "optimal" number of clusters, i.e. the alphabet size of SAX. The idea is finding the number of clusters that stops sharp dropping of the WSS (within cluster sum of squares), which is illustrated in Figure 2.

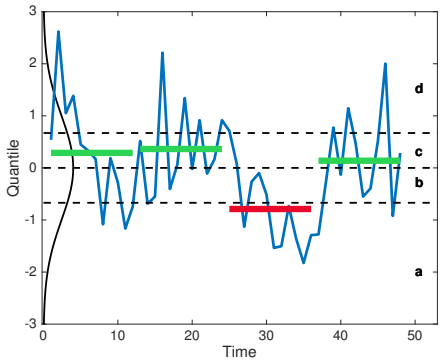

Figure 1: SAX labeling of time series data. The dashed lines indicate discretization boundaries.

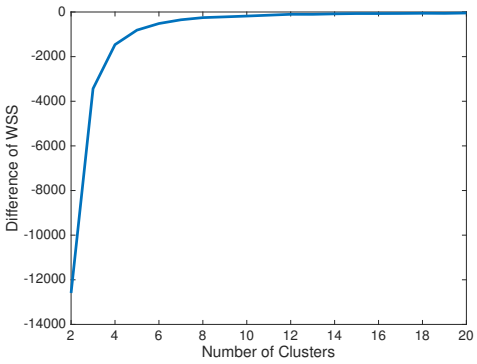

Figure 2: WSS difference versus number of clusters in training data. We select 8 as a good number of clusters.

2.2 Stationarity and Drift Model

Many time series in practice, such as the economic process and the wind speed, are difficult to predict since they are not stationary. Intuitively, the statistical properties of these processes, such as mean and variance, vary over time [1]. Logarithm and differencing are two widely used preprocessing methods for non-stationary time series [1]. The logarithm is useful to stabilize the variance of a time series of which larger values tend to have larger variance, meanwhile it helps to expand the difference of small values around zero. Differencing (1-st order derivative), i.e. computing the differences between consecutive observations, is useful to stabilize the mean of a time series by removing changes in the level of a time series, and so eliminating trend and seasonality. Assume that the original data of length N is $\mathbf{X} = [x_0, x_1, \cdots, x_{N-1}]$, and our goal is to get a drift model,

$$x_t - x_{t-1} = \hat{c} + e_t \tag{1}$$

where $\hat{c}$ is our estimated mean value of the drift, and e_t is assumed as white noise. Unlike the conventional time series models that directly take all the historic difference

Table 1: Global SAX guards for the wind speed prediction task, values are in m/s.

Symbol	a	b	c	d	e	f	g	h
Guard	(-∞, 0.59)	[0.59, 1.16)	[1.16, 1.58)	[1.58, 1.96)	[1.96, 2.34)	[2.34, 2.76)	[2.76, 3.33)	[3.33, +∞)

Table 2: k-means centroids for the wind speed prediction task, values are in m/s.

Symbol	a	b	c	d	e	f	g	h
Centroid	0.76	1.22	1.68	2.20	2.82	3.63	4.81	7.46

values into account to estimate $\hat{c}$, our syntactic model discovers patterns sharing similar behaviors to individually get the estimations of $\hat{c}$. Once $\hat{c}$ is learned from training data, Equation 1 is also used for forecasting with known previous value.

Apart from global SAX and differencing, we also investigate the following strategies of preprocessing, and compare the results in the experiment (see Section 4.3).

- **local SAX** aggregates, discretizes, and normalizes data in each sliding window, see [18] for details.
- **k-means** with the identical alphabet size as SAX is listed in Table 2, which shows the centroids of the symbols obtained in experimental case study one (see Section 4.3). All numeric values are abstracted to the symbol with the closest associated centroid.
- **logarithm differencing** compute the logarithm difference between consecutive observations, which actually reflects the ratio relations.

3. MODEL LEARNING

3.1 Regression Automata

We provide a very concise description of DFAs, the reader is referred to [21] for a more elaborate overview. A *deterministic finite automaton (DFA)* is a quadruple $A = \langle Q, T, \Sigma, q_0 \rangle$ where Q is a finite set of states, $T : (Q, \Sigma) \to Q$ are labeled transitions with labels coming from an *alphabet* Σ, $q_0 \in Q$ is the start state. A DFA computation starts in the start state q_0 and traverses transitions according to a given input string (sequence) $s_1 \dots s_n \in \Sigma^*$. At every index $1 \leq i \leq n$, the current state of the DFA is changed from source state q_{i-1} to target state $T(q_{i-1}, s_i)$. This computation is called deterministic because there exists exactly one target for every source-symbol pair. In contrast to the commonly used HMMs [22], the computation path of a given DFA is thus completely determined for a given input string. This property makes them easier to learn. Learning DFAs is however much harder than learning Markov chains because (like HMMs) the traversed states are unknown (hidden) when given only input data.

A regression automaton (RA) is a quintuple $A = \langle Q, T, \Sigma, q_0, P \rangle$ where $\langle Q, T, \Sigma, q_0 \rangle$ is a DFA, and P is a prediction function $P : Q \to \mathbb{R}$. The prediction function assigns a prediction value to every state $q \in Q$. The computation of an RA is identical to that of a DFA, any numeric input data is ignored. Whenever a computation is in a state q, the value $P(q)$ is only used as a prediction for the next numeric data value. In our case, we use the preprocessing described above

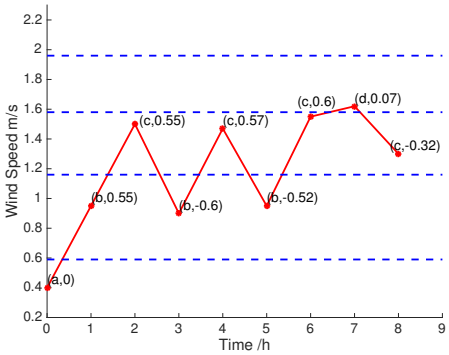

Figure 3: Our labeling of time series data, consisting of symbols and difference values. The dashed lines indicate discretization boundaries (using code book in Table 1). To avoid redundancy of data, the values in this plot have been aggregated by SAX.

to obtain discretized symbols based on a time series signal, and numeric values based on the difference of the series, see Figure 3. The state of an RA is thus fully determined by the syntactic data, and the predicted drift value only depends on the current state. RAs can be seen as mappings from symbolic sequences to drift values.

3.2 Evidence-Driven State-Merging

The current state-of-the-art in DFA learning is evidence-driven state-merging in the red-blue framework (EDSM) [23], possibly with some search procedure (see, e.g., [19]) in order to continue searching once a possible local optimum has been reached. In the following, we briefly explain the main steps of this algorithm together with our adaptations needed to handle numeric data.

3.2.1 Prefix Tree Construction

The first step in EDSM is to build a Prefix Tree (PT) from the training data. For each input sample w from the training data, a chain is created by introducing a state between each letter w_i $(1 \leq i \leq |w|)$. This chain is inserted into the PT by traversing its labeled transitions until the word is fully inserted, or a leaf is reached. Upon reaching the leaf, the remaining sequence is appended at this position. For every state q in a PT, there exists exactly one computation that reaches q. A PT therefore encodes exactly the information in the (syntactic) training data, without any generalization. The set of states Q is extended to contain a null state $q_\perp$, to represent transitions for which no input data exists in the training sample, i.e., $T(q, l) = q_\perp$ means it is currently unknown what the target state is from state q with label l.

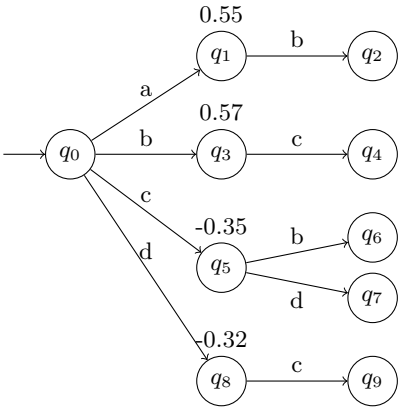

Figure 4: First two levels of the APTA generated by sliding a window of length two over the data in Figure 3. Between each data point a state is created, and data points itself are used to label transitions. Each node stores the mean of the differences of its outgoing transitions: e.g. for node q_5, there are three outgoing transitions stemming from three input samples $(c, 0.55)(b, \text{-}0.6), (c, 0.57)(b, \text{-}0.52), (c, 0.6)(d, 0.07)$ creating this branch of the tree. The value -0.35 is the mean value of $\text{-}0.6, \text{-}0.52, +0.07$.

For RAs, the PT structure is constructed in the standard way using only the syntactic data, see Figure 4 for an example. The transitions are labeled with the symbol corresponding to the chosen discretization. In addition to the prefix tree structure, we aggregate the numeric values of all outgoing transitions in each node; the numeric values above states q_1, q_3, q_5 and q_8 are the average values of the differences of all outgoing transitions. If we want to predict the next value following 1.3, i.e. the original value of last data in Figure 3. we follow the transitions with the corresponding symbolic label, e.g., c, from the starting state q_0. In our example, it will transition to state q_5. By applying the reverse translation from Equation 1, the predicted value is $1.3 - 0.35 = 0.95$.

3.2.2 Merging States in EDSM

The PT, encoding all the training data without generalization, usually leads to high variance models sensitive to noise, and has an increased risk of overfitting. The goal of *DFA learning* is to find a *smallest* DFA A that is *consistent* with the training data set [24]. Seeking this DFA is an active research topic in grammatical inference, see [25]. The PT is iteratively made smaller by heuristically *merging* pairs of states (q, q'), and re-estimating the transition function (matrix) T. Every such merge creates a new state q'' that has the incoming and outgoing transitions of both q and q'. The merged states q and q' are removed from the model. When a merge introduces a non-deterministic choice, i.e., $T(q, a) = q_1$ and $T(q', a) = q_2$ both exist for some label a, states q_1 and q_2 are merged as well. This is called the determinization process. Which merge to perform is determined using a heuristic (typically an evidence measure). Standard EDSM, for instance, maximizes the total number of merged states with matching outputs [23]. Probabilistic DFAs can be learned using statistical distances such as KL-divergence [26] or outcomes of for instance likelihood ratio tests [27].

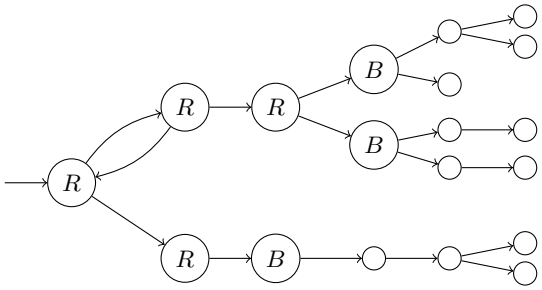

Figure 5: Red-Blue Framework: Starting at the root, the algorithm tries to find the smallest consistent state machine. Already identified parts of the target are marked *red*, and direct neighbors of those states as *blue*. The heuristic focuses on the fringe of the marked states, instead of having to check all possible combinations of states.

In DFASAT and in this paper, the widely used *red-blue framework* [23] is applied for guiding the merge process. As shown in Figure 5, the red-blue framework only merges red $r \in R \subseteq Q$ and blue $b \in B \subseteq Q$ states. The red states and the transitions between them form the currently constructed DFA, the blue states are still to be identified transitions, potentially to new states of the DFA. The new state q'' resulting from a red-blue merge is colored red, i.e., $R := R \cup \{q''\}$. In addition, every non-red target state $q \in Q \setminus R$ that is the target of a transition $T(r, l) = q$, for any $l \in \Sigma$, with a red source state $r \in R$, is colored blue, i.e., $B := B \cup \{q\}$. In this way, the framework maintains a core of red states with a fringe of blue states (see Figure 5). Initially, the start state of the APTA is colored red, and its children (targets for every symbol) are colored blue.

Merges are only allowed if the resulting DFA is still *consistent*, e.g., states with different outputs cannot be merged [23], states with significantly different outgoing transition labels cannot be merged [19], or states with significantly different outgoing transition label distributions cannot be merged [28]. Overall, the run time complexity of red-blue algorithms is bounded by $|S| \cdot n$, where S is the input set and n the size of the final model [23]. For the RA learning problem, new heuristics and consistency tests are needed because the goal is to produce accurate numeric predictions instead of accurate predictions of syntactic input/output values.

3.2.3 Merging for Regression Automata

Instead of the statistical or input/output consistency checks in traditional state merging approaches described before, we allow merges between states q and q' where the mean value of difference is smaller than a given threshold. Take the data series in Figure 3 for example, patterns "ab" and "bc" share a similar trend, i.e., similar difference values stored in q_1 and q_3 in Figure 4. We only consider merges in which all states that are merged due to determinization have sufficiently similar difference values. In addition to these difference values, we also store the number of occurrences in every state.

To evaluated possible merges and choose the best merge, we use the variant Akaike information criterion (AIC) for regression models [29] as a merge heuristic:

$$\Delta AIC = 2\left(\kappa_{before} - \kappa_{after}\right) + n \lg \frac{RSS_{before}}{RSS_{after}} \quad (2)$$

where κ_{before} and κ_{after} is the number of parameters in the model, i.e., the number of states before and after the merge respectively, n is the number of data points in training set for fitting the model, RSS_{before} and RSS_{after} are the residual errors, i.e., the total square error in states before and after merge models. We compute AIC difference in each iteration of merge, there could exist more than one pairs of red-blue states, i.e. candidates for merge, however, only the highest AIC difference of candidate pair is selected for merge to improve model performance most significantly. An overview of our new state merging algorithm is given in Algorithm 1, where $\#\mathsf{occ}(q)$ denotes the number of occurrences in state q.

3.3 Model Smoothing

Another source of difficulty in applying syntactic models to regression tasks is model smoothing. Taking the model in Figure 4 for instance, it can happen that new data contains a symbol "e". For this case, no matching transition exists, and it is impossible to obtain a prediction from the model. In this paper, we solve this problem using a relatively simple strategy: we follow the transition with the symbol closest to the input "e" according to the discretization scheme. In this example, state q_8 is reached by following the transition for symbol "d". In this way it is possible to make a numeric prediction even for sequences that were neither seen in training data nor generalized to during learning. In our case studies, this only happens less than 0.1% of the time.

3.4 Sliding Window Length

One of the key problems in our learning task is to determine the length of the sliding window, i.e., how many historical data points the prediction would rely on. Figure 6 illustrates the relationship between fitting error and model complexity for the wind speed training data used in the experiments. Larger length of sliding window results in more layers in PT and hence more states. E_{in} and E_{out} are the fitting mean square error in training data and testing data respectively. We can see that by increasing the model complexity (sliding window length), E_{in} decreases sharply, while E_{out} becomes increasingly worse, which is typically the result of overfitting. In practice, we favor simpler models in order to reduce the risk of overfitting. The models, of which window length are less than 5, have relative small E_{out}. We fix the length as 4 for the main experiments, and also try length 8 in order to discover whether state merging can overcome the drop in E_{out}, see Section 4.4

4. EXPERIMENTS

4.1 Typical Methods for Comparison

In this paper, regression automata are compared with other widely used prediction models.

- **Persistent Model** is the most widely used baseline in time series forecasting tasks, which just let the predicted value equal its preceding known one.
- **Autoregressive Integrated Moving Average (ARIMA)** To ensure fairness when comparing prediction results, we use integrated ARMA (ARIMA) in this paper since as we apply 1-st order derivatives in the preprocessing procedure. The maximum order of AR and MA is fixed to 3 since we have sliding window of length 4. We select the "best fitting model" with with lowest AIC and highest log-likelihood.
- **Recurrent Neural Network (RNN)** using long-term short-term nodes [30] were very successful. We train a model on normalized differences input and output. We select 3 layers and 15 hidden neurons. The output function is ReLU.
- **Regression Tree (RT)** is a *IF-THEN* rules based model, which been applied successfully in time series forecasting [31]. In this paper, the regression tree is built using scikit-learn *DecisionTreeRegressor* tool[1], which is based on CART algorithm [32].

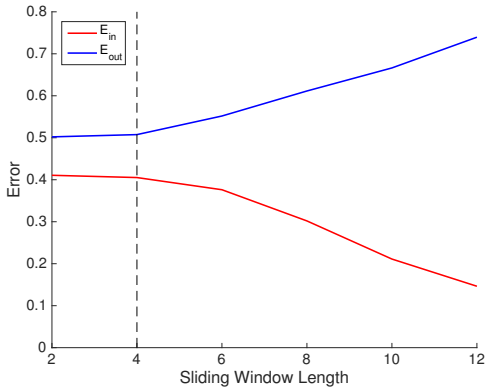

Figure 6: PT Fitting Error vs Window Length: Errors E_{in}, E_{out} on training data and testing data calculated on the PT, the starting data structure for the learning algorithm.

4.2 Evaluation Metrics

For notational convenience, we collect all the predicted data and form a new vector $\hat{\mathbf{v}} = [\hat{v}_1, \hat{v}_2, \cdots, \hat{v}_k, \cdots, \hat{v}_N]$. The corresponding vector of actual values is defined as $\mathbf{v} = [v_1, v_2, \cdots, v_k, \cdots, v_N]$. In this paper, the following types of indices are calculated for fair comparisons:

- Root mean square error:

$$RMSE = \sqrt{\frac{1}{N}\sum_{k=1}^{N}\left(\hat{v}_k - v_k\right)^2} \quad (3)$$

- Mean absolute percentage error:

$$MAPE = \frac{1}{N}\sum_{k=1}^{N}\left|\frac{\hat{v}_k - v_k}{v_k}\right| \times 100\% \quad (4)$$

- Mean absolute error:

$$MAE = \frac{1}{N}\sum_{k=1}^{N}\left|\hat{v}_k - v_k\right| \quad (5)$$

[1]http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

Algorithm 1 State-merging for Regression Automata

Require: an input sample S, an occurrence threshold t, and a difference threshold t_d
  $A = \mathsf{PT}(S)$ ▷ construct the prefix tree
  $R = \{q_0\}$ ▷ color the start state red
  $B = \{q \in Q \setminus R \mid \exists l \in \Sigma : T(q_0, l) = q\}$ ▷ color all its children blue
  while $B \neq \emptyset$ **do** ▷ while A contains blue states
    if $\exists b \in B$ s.t. $\forall r \in R$ holds $merge(A, r, b, t_d) =$ FALSE **then** ▷ if a blue state is inconsistent with all red states
      $R := R \cup \{b\}$ ▷ color b red
      $B := B \cup \{q \in Q \setminus R \mid \exists l \in \Sigma : T(q, l) = q$ and $\#\mathsf{occ}(q) \geq t\}$ ▷ color all its children with at least t occurrences blue
    else
      for all $b \in B$ and $r \in R$ **do** ▷ forall red-blue pair of states
        compute the ΔAIC of $\mathsf{merge}(A, r, b)$ ▷ find the best performing merge
      end for
      call the $\mathsf{merge}(A, r, b, t_d)$ with highest ΔAIC ▷ perform the best merge
      let q'' be resulting state
      $R := R \cup \{q''\}$ ▷ color the resulting state red
      $R := R \setminus \{r\}$ ▷ uncolor the merged red state
      $Q := Q \setminus \{r, b\}$ ▷ remove the merged states
      $B := \{q \in Q \setminus R \mid \exists r \in R, l \in \Sigma : T(q, l) = q$ and $\#\mathsf{occ}(q) \geq t\}$ ▷ recompute the set of blue states
    end if
  end while
  return A

Algorithm 2 Merging two regression states: merge (A, q, q', t_d)

Require: an RA $A = \langle Q, T, \Sigma, q_0, P\rangle$, two states $q, q' \in Q$, and a threshold t_d
Ensure: if q and q' are inconsistent, return FALSE; else return A with q and q' merged.
  if $|P(q) - P(q')| \geq t_d$, **then return** FALSE ▷ return FALSE if q is inconsistent with q'
  create a new state q'', and set $Q := Q \cup q''$ ▷ add a new state q'' to A
  set $\#\mathsf{occ}(q'') := \#\mathsf{occ}(q) + \#\mathsf{occ}(q')$ and $P(q'') := \frac{\#\mathsf{occ}(q)P(q) + \#\mathsf{occ}(q')P(q')}{\#\mathsf{occ}(q'')}$ ▷ (update $\#\mathsf{occ}$ and P)
  for all symbols $l \in \Sigma$ **do** ▷ for all transitions from q and q'
    if $T(q, l)\neg = q_\perp$ **then** set $T(q'', l) := T(q, l)$ ▷ copy outgoing transitions from q
    if $T(q, l)\neg = q_\perp$ **then** set $T(q'', l) := T(q', l)$ ▷ copy outgoing transitions from q'
  end for
  for all states $q_s \in Q$ and symbols $l \in \Sigma$ such that $T(q_s, l) \in \{q, q'\}$ **do** ▷ for all source states of transitions to q or q'
    set $T(q_s, l) := q''$ ▷ copy incoming transitions to q or q'
  end for
  for all symbols $l \in \Sigma$ **do** ▷ for all old transitions from q and q'
    if $T(q, l)\neg = q_\perp$ and $T(q', l)\neg = q_\perp$, **then** res := $\mathsf{merge}(A', T(q, l), T(q', l), t_d)$ ▷ determinize the targets
    if res equals FALSE, **then return** FALSE and undo the merge ▷ return FALSE if the targets are inconsistent
  end for
  return true

4.3 Experiment Results

4.3.1 Case Study One: Wind Speed Prediction

The data used in this case is from the online weather database of Delft University of Technology[2]. There is data from 16 weather stations in total. We selected station "Rijnhaven" among the stations with longest observation period, from 2013-04-23 to 2015-10-12. We calculate hourly averages of the wind speed, and predict one hour ahead. Using a sliding window of 4 hours, the data was split into a training set containing 17537 windows with 70148 data points, and a test set containing 4113 windows with 16452 data points.

To begin with, we compare different preprocessing strategies in prefix tree. SAX generally outperforms k-means, which provides the insights that in the wind data, the symbolization based on equal space of probability better discovers the patterns for the drift estimation. Logarithm differencing generally helps to get lower MAPE, because it reflects ratio relationship, which is consistent with the definition of MAPE. Though local SAX is powerful in anomaly pattern discovery, see [18], global SAX makes more sense in the experiment. The global SAX and differencing strategies are chosen in the following cases studies. To make a fair comparison, all other baselines are fed with difference inputs.

The evaluation results of different models are summarized in Table 4, where the best for each index is in bold. Our model outperforms all other baselines with in MAPE while ARIMA shows slightly better results in RMSE and MAE.

The final merged state machine is illustrated in Figure 7. The model's size is drastically reduced from 350 states to 20 states (except the start state and the leaf states since they are useless for the regression). The top-most state is the start state. Starting from this state, the model moves along transitions by first discretizing the next time series value and then following the transition with that discretized label. The first value in every state (a circle) is the mean

[2]http://weather.tudelft.nl/csv/

Table 3: Comparisons of Different Preprocessing Strategies

Methods	Gloabl-SAX-diff	k-means-diff	Local-SAX-diff	Global-SAX-logdiff	k-means-logdiff	Local-SAX-logdiff
RMSE (m/s)	**0.5031**	0.6501	0.5115	0.5072	0.6211	0.5124
MAPE (%)	18.7711	25.3068	18.9490	**18.3330**	20.6989	18.7300
MAE (m/s)	**0.3660**	0.4850	0.3725	0.3666	0.4347	0.3722

value of difference values from the training data reaching that state. These are used to make predictions. The second values shows the number of occurrences of every state.

The automaton has 11 loops, i.e., transitions where origin and target state are the same, which are introduced by state merge. Given the historic data that already abstracted into the pattern *abc* (continuously increasing wind speed) for instance, it starts from root state and reach the state 0.054, which means it is expected to drift up 0.054 m/s. And for the pattern *hgf*, it reaches the state -0.107, which predicts a 0.107 m/s drift down. The pattern *hhh* staying high speed for 3 hours, reaches -0.145 and is predicted to slope down.

4.3.2 Case Study Two: Multi-Step Prediction

In this case study, we evaluate the regression models for multi-steps, i.e., more than one-hour-ahead forecasting, still using the data sets with one data point per hour. Our input data again consists of windows, pre-processed as in the previous case studies, except for the last element of the window being the value for multiple steps ahead. For example to predict a value three hours into the future, at time $T+3$, our training data contains windows of the form $(x_{T-2}, x_{T-1}, x_T, x_{T+3})$. The evaluation results of 3-hour-ahead and 6-hour-ahead predictions are listed in Table 5 and Table 6. With the increasing of prediction interval, the persistent model doesn't work so well as in *Case One*. Our model improves significantly compared to other approaches.

4.3.3 Case Study Three: Wind Power Prediction

In this case study, we investigate the wind power prediction using the data set from National Renewable Energy Laboratory (NREL) of U.S. Department of Energy[3]. The training data starts from *2004-01-02 00:00:00* to *2006-05-31 23:50:00*, while the testing data starts from *2006-06-01 00:00:00* to *2007-01-01 23:50:00*.

Similar to the wind speed forecasting case study, we apply our model in wind power prediction, i.e. using the historical wind power data as input and the one, three, and six hour ahead power as output. Wind power forecasting is challenging due to the non-linearity resulting from the dead zone and the saturation characters. More specifically, power output has zero value when the wind speed value is lower than the wind turbines' cut-in threshold, meanwhile the output reaching constant rated power if the wind speed is greater than the cut-off upper-bound. Table 7 gives a comparison of the power prediction for different models. Note that due to the dead zone character of wind power system, lots of zero value of real data exist making MAPE metric ill-defined. Only RMSE and MAE are reported for comparison. From the results we can see that ARIMA performance better in 1-hour-ahead data set. ARIMA is powerful in one step ahead because the on-line updating of both input autoregressive values and residual errors is efficient in short term forecasting. However, in relative longer prediction intervals, our model gains improvement over baselines.

[3]http://www.nrel.gov/electricity/transmission/eastern_wind_dataset.html

4.4 Learning and Model Complexity

Learning finite state automata exactly with incomplete samples is NP-hard [33]. State-merging algorithms use heuristics, and generally have a worse-case complexity in the order of a cubic term in the input data size. Evaluating a regression automata is linear sequence of looking up the transitions to the last node, and adding the predicted speed difference to the previous speed value. Our automata only have about 20 states, requiring to store 20 float values and at most $20 \times |\Sigma|$ triples of state-symbol-state for the transition matrix. In practice, the runtime of RAs, including training and testing, on our Intel 2.6 GHz i5 processors using a single core doesn't need more than a minute. The comparisons with all baselines are listed in Table 9. We also compare the performance of the prefix tree with the performance our merged regression automata. The prefix tree is a compact representation of the input data and is generated in linear time. While it is generated much faster, it does not generalize, and is large in size. In Figure 6, shows the training and testing error in prefix trees with different depths. The longer the window size, i.e. the higher the order of auto-regression, the deeper the prefix tree will get. We try to investigate how state merging influences the model performance and how it relates to varying size measured in states. Table 8 shows the benefit of the learning process. $impr(\%)$ is the automaton's improvement in RMSE over prefix trees. For longer sliding windows, state merging clearly improves the RAs performance more. The RMSE of model with length-8 has very closed accuracy with length-4 model after learning. It surprisingly provides the evidence of the generalization efficiency of our learning algorithm.

5. CONCLUSION

The main contribution of this work is the extension of automata for time series regression. A novel state merging approach for learning small automata from numeric data is proposed using the DFASAT framework. To the best of our knowledge, we provide the first automaton model together with a learning algorithm that can be directly applied to time series regression problems. Several case studies are performed, which demonstrate that our approach allows for powerful generalization from training to testing data. In addition to good performance in practice, our algorithm provides succinct and interpretable models, which can be essential for deployment in real wind power parks. In the near future, we will make even more use of the numeric wind speed/power data during merging. This way, we can exploit spatial information, either by modifying our preprocessing to create a multivariate regression problem, or considering additional information such as location, directionality, cor-

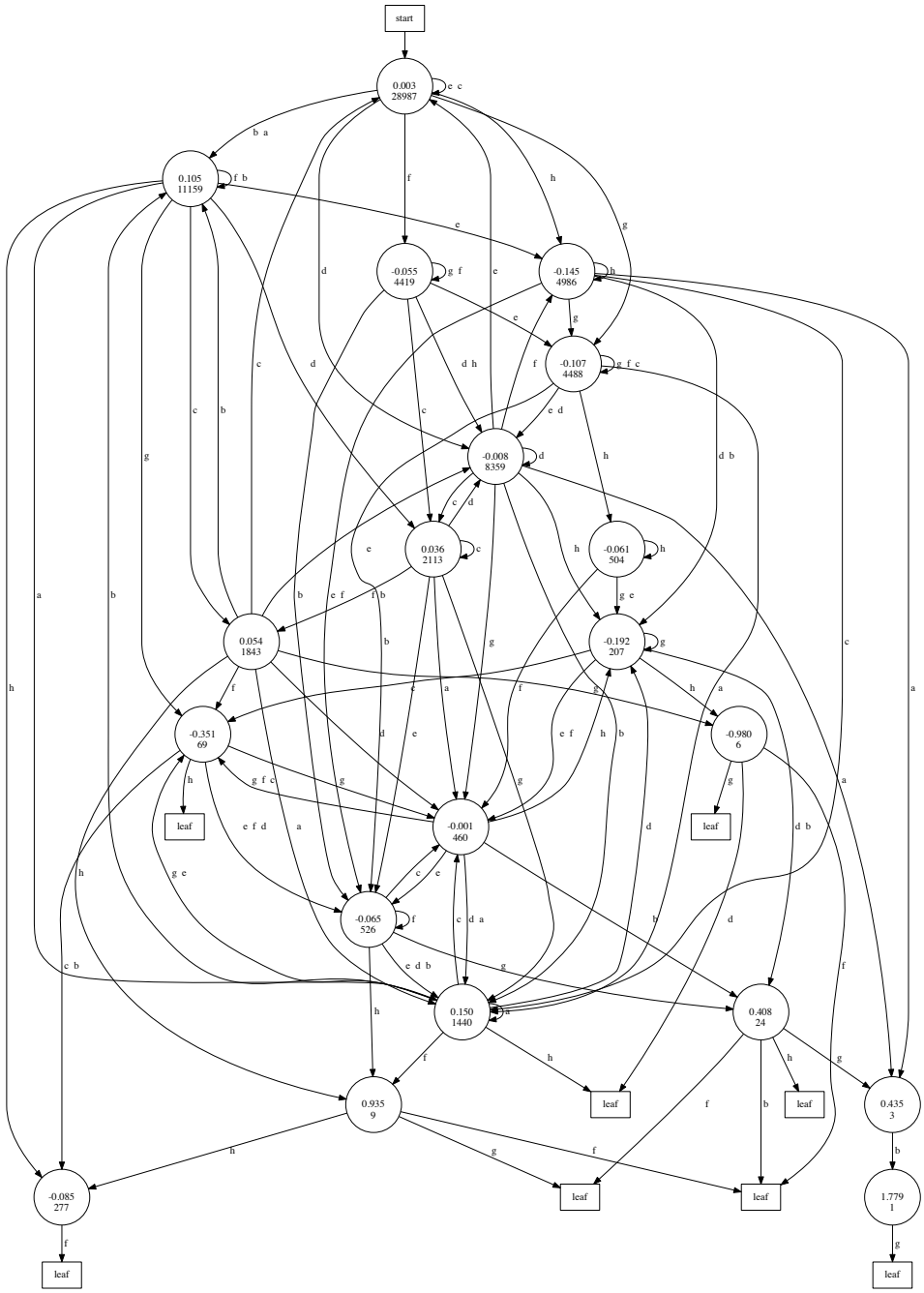

Figure 7: The merged RA for the one-hour-ahead wind-speed prediction.

Table 4: One-hour-ahead Speed Prediction Performance Comparisons.

Model	RA	Prefix Tree	ARIMA	RNN	RT	Persistence
RMSE (m/s)	0.4996	0.5031	**0.4956**	0.6060	0.6884	0.5077
MAPE (%)	**18.5797**	18.7711	18.7355	24.483	27.1475	18.6090
MAE (m/s)	0.3629	0.3660	**0.3615**	0.4707	0.5116	0.3685

relation, and standard deviations during consistency checks and merging. Additionally, different discretization strategies could be further invested for better abstraction of numerical data. An interesting approach would be to discretize this data on-the-fly during the learning process, as has been before with temporal data in timed automata [27]. In ad-

Table 5: 3-hour-ahead Speed Prediction Performance Comparisons.

Model	RA	Prefix Tree	ARIMA	RNN	RT	Persistence
RMSE (m/s)	**0.8722**	0.8753	0.8821	1.0015	0.9892	0.8930
MAPE (%)	**32.5249**	32.6794	33.1649	37.2406	38.8493	33.2933
MAE (m/s)	**0.6321**	0.6347	0.6432	0.7637	0.7404	0.6489

Table 6: 6-hour-ahead Speed Prediction Performance Comparisons.

Model	RA	Prefix Tree	ARIMA	RNN	RT	Persistence
RMSE (m/s)	**1.2048**	1.2083	1.2286	1.2617	1.3038	1.2344
MAPE (%)	**46.8085**	47.0155	48.0161	47.02642	51.9327	48.1143
MAE (m/s)	**0.8974**	0.9013	0.9192	0.9444	0.9855	0.9226

Table 7: Power Prediction Performance Comparisons.

	Model	RA	Prefix Tree	ARIMA	RNN	RT	Persistence
1-hour-ahead	RMSE (MW)	1.8952	1.8979	**1.8673**	1.9859	2.6541	1.9830
	MAE (MW)	1.2610	1.2613	**1.2312**	1.2814	1.8066	1.2793
3-hour-ahead	RMSE (MW)	**3.7427**	3.7435	3.7738	4.6883	4.4193	3.8796
	MAE (MW)	**2.6438**	2.6458	2.6196	3.6595	3.1597	2.6832
6-hour-ahead	RMSE (MW)	**5.0053**	5.0088	5.0434	5.1567	5.4872	5.1486
	MAE (MW)	**3.6529**	3.6546	3.6540	3.7355	4.0661	3.6529

Table 8: Improvement due to state-merging over the prefix tree in the RSME measure at different sliding window length.

	1-hour-ahead			3-hour-ahead			6-hour-ahead		
	RA	Prefix Tree	impr (%)	RA	Prefix Tree	impr (%)	RA	Prefix Tree	impr (%)
length-4	0.4996	0.5031	0.70	0.8722	0.8753	0.35	1.2048	1.2083	0.29
length-8	0.4994	0.5959	16.19	0.8737	0.9333	6.39	1.2089	1.2495	3.25

Table 9: Runtime Comparisons.

Model	RA	Prefix Tree	ARIMA	RNN	RT	Persistence
Runtime	19.086s	1.806s	1m48.796s	19m54.580s	2.035s	1.081s

dition to mean forecasting, probabilistic prediction is also important for decision purpose [34]. RAs can generate the probabilities for symbolic forecasts, which will be done in the near future. We will also try the rolling evaluation for concept drift problems [17].

6. ACKNOWLEDGMENTS

This work was partially supported by Technologiestichting STW VENI project 13136 (MANTA) and NWO project 62001628 (LEMMA), as well as FNR Luxembourg AFR project PAULINE.

References

[1] Jonathan Cryer and 2008 Kung-sik Chan. *Time Series Analysis with Applications in R*. Springer, 2008.

[2] Jan G De Gooijer and Rob J Hyndman. 25 years of time series forecasting. *International journal of forecasting*, 22(3):443–473, 2006.

[3] Ma Lei, Luan Shiyan, Jiang Chuanwen, Liu Hongling, and Zhang Yan. A review on the forecasting of wind speed and generated power. *Renewable and Sustainable Energy Reviews*, 13(4):915–920, 2009.

[4] A Tascikaraoglu and M Uzunoglu. A review of combined approaches for prediction of short-term wind speed and power. *Renewable and Sustainable Energy Reviews*, 34:243–254, 2014.

[5] Alexandre Costa, Antonio Crespo, Jorge Navarro, Gil Lizcano, Henrik Madsen, and Everaldo Feitosa. A review on the young history of the wind power short-term prediction. *Renewable and Sustainable Energy Reviews*, 12(6):1725–1744, 2008.

[6] Gregor Giebel, Richard Brownsword, George Kariniotakis, Michael Denhard, and Caroline Draxl. The state-of-the-art in short-term prediction of wind power: A literature overview. Technical report, ANEMOS. plus, 2011.

[7] Jose Luis Torres, Almudena Garcia, Marian De Blas, and Adolfo De Francisco. Forecast of hourly average wind speed with ARMA models in Navarre (Spain). *Solar Energy*, 79(1):65–77, 2005.

[8] EA Bossanyi. Short-term wind prediction using Kalman filters. *Wind Engineering*, 9(1):1–8, 1985.

[9] RA Schlueter, G Sigari, and A Costi. Wind array power prediction for improved operating economics and reliability. *Power Systems, IEEE Trans. on*, 1(1):137–142, 1986.

[10] Zhenhai Guo, Weigang Zhao, Haiyan Lu, and Jianzhou Wang. Multi-step forecasting for wind speed using a modified EMD-based artificial neural network model. *Renewable Energy*, 37(1):241–249, 2012.

[11] MA Mohandes, TO Halawani, S Rehman, and Ahmed A Hussain. Support vector machines for wind speed prediction. *Renewable Energy*, 29(6):939–947, 2004.

[12] Ioannis G Damousis, Minas C Alexiadis, John B Theocharis, and Petros S Dokopoulos. A fuzzy model for wind speed prediction and power generation in wind parks using spatial correlation. *Energy Conversion, IEEE Transactions on*, 19(2):352–361, 2004.

[13] John E Albus, RH Anderson, JM Brayer, R DeMori, H-YF Feng, SL Horowitz, B Moayer, T Pavlidis, WW Stallings, PH Swain, et al. *Syntactic pattern recognition, applications*, volume 14. Springer Science & Business Media, 2012.

[14] Iain L MacDonald and Walter Zucchini. *Hidden Markov and other models for discrete-valued time series*, volume 110. CRC Press, 1997.

[15] Ahmet D Sahin and Zekai Sen. First-order markov chain approach to wind speed modelling. *Journal of Wind Engineering and Industrial Aerodynamics*, 89(3):263–269, 2001.

[16] Guglielmo D'Amico, Filippo Petroni, and Flavio Prattico. Wind speed and energy forecasting at different time scales: A nonparametric approach. *Physica A: Statistical Mechanics and its Applications*, 406:59–66, 2014.

[17] C Lee Giles, Steve Lawrence, and Ah Chung Tsoi. Noisy time series prediction using recurrent neural networks and grammatical inference. *Machine learning*, 44(1-2):161–183, 2001.

[18] Jessica Lin, Eamonn Keogh, Li Wei, and Stefano Lonardi. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and knowledge discovery*, 15(2):107–144, 2007.

[19] Marijn JH Heule and Sicco Verwer. Software model synthesis using satisfiability solvers. *Empirical Software Engineering*, 18(4):825–856, 2013.

[20] Cyril Goutte, Peter Toft, Egill Rostrup, Finn Å Nielsen, and Lars Kai Hansen. On clustering fMRI time series. *NeuroImage*, 9(3):298–310, 1999.

[21] Thomas A. Sudkamp. *Languages and Machines: an introduction to the theory of computer science*. Addison-Wesley, third edition, 2006.

[22] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[23] Kevin J Lang, Barak A Pearlmutter, and Rodney A Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, pages 1–12. Springer, 1998.

[24] Dana Angluin. Inductive inference of formal languages from positive data. *Information and control*, 45(2):117–135, 1980.

[25] Sicco Verwer, Rémi Eyraud, and Colin De La Higuera. Pautomac: a probabilistic automata and hidden markov models learning competition. *Machine learning*, 96(1-2):129–154, 2014.

[26] Franck Thollard, Pierre Dupont, Colin de la Higuera, et al. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *ICML*, pages 975–982, 2000.

[27] Sicco Verwer, Mathijs de Weerdt, and Cees Witteveen. A likelihood-ratio test for identifying probabilistic deterministic real-time automata from positive data. In *Grammatical Inference: Theoretical Results and Applications*, pages 203–216. Springer, 2010.

[28] Rafael C Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In *Grammatical Inference and Applications*, pages 139–152. Springer, 1994.

[29] Kenneth P Burnham and David R Anderson. *Model selection and multimodel inference: a practical information-theoretic approach*. Springer Science & Business Media, 2002.

[30] Felix A Gers, Douglas Eck, and Jürgen Schmidhuber. Applying lstm to time series predictable through time-window approaches. In *Artificial Neural NetworksâĂŤICANN 2001*, pages 669–676. Springer, 2001.

[31] A Troncoso, S Salcedo-Sanz, C Casanova-Mateo, JC Riquelme, and L Prieto. Local models-based regression trees for very short-term wind speed prediction. *Renewable Energy*, 81:589–598, 2015.

[32] L Breiman, JH Friedman, R Olshen, and CJ Stone. Classification and regression trees. 1984.

[33] Dana Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39(3):337–350, 1978.

[34] Pierre Pinson et al. Wind energy: Forecasting challenges for its operational management. *Statistical Science*, 28(4):564–585, 2013.