Evaluating Extended Automata for Sequential Regression

A comparison against standard baselines

Job Vonck[1]
Supervisor: S.E. Verwer[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 21, 2026

Name of the student: Job Vonck
Final project course: CSE3000 Research Project
Thesis committee: S.E. Verwer, K. Atasu

An electronic version of this thesis is available at https://repository.tudelft.nl/.

Abstract

Sequential regression has many real-world applications, including energy consumption forecasting and traffic prediction. In this paper, the use of extended regression automata for sequential regression is investigated. The automata are learned using an MSE-based approach with the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) to score merge and split operations, where the use of BIC is a novel contribution of this work. The approach is evaluated on three datasets and compared against persistence, a regression tree, and a random forest. The automata perform comparably on the small dataset, but on the larger datasets, they do not outperform the regression tree and random forest baselines. This is mainly caused by a bias towards splitting during learning, which prevents the automata from making use of loops and causes them to resemble trees. We further found that BIC produced smaller automata and ran faster, but its accuracy was mixed relative to AIC for our datasets.

1 Introduction

Sequential regression is a problem encountered in many real-world domains, including energy consumption forecasting [1], traffic prediction [2], and others. Many of these problems can naturally be modelled using states and transitions. This is especially attractive for repeated behaviour, since automata can model this using loops, while tree-based models must keep expanding their structure to capture this. Another benefit is that automata are interpretable by design, which is valuable in domains where understanding the model's decisions can be as important as accuracy.

Automata are one of the pillars of computer science. They are often used to model the behaviour of software systems. However, they are still limited in their expressiveness. Extended automata extend this behaviour by adding values and guards. These allow for more fine-grained control, such as modelling software executions [3].

The use of automata for regression tasks has been studied before [4]. The authors introduced a new type of automaton called a regression automaton, which requires clustering the input values into buckets and assigning a symbol to each bucket. This approach is effective, but it requires an additional step before the automaton can be learned. This also introduces extra parameters, such as the alphabet size. Our work avoids this step by using extended automata, which operate directly on the values with guards.

To learn these automata, we make use of FlexFringe [5]. FlexFringe is an adaptable framework that supports multiple algorithms. In our approach, we make use of both the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) to score the split/merge operations. The use of BIC in this setting is one of the novel contributions of this work.

We evaluate this approach on three different datasets with different levels of complexity and compare it to persistence, a regression tree, and a random forest.

Our main research questions are:

- RQ1: Can extended automata outperform baseline regression methods for sequential regression?
- RQ2: Do extended automata learned with BIC produce better results than those learned with AIC?

This paper is organized as follows. Section 2 introduces extended finite state machines, extended regression automata, state-merging/splitting, FlexFringe, and information criteria.

Section 3 describes how the data was preprocessed and how the automata were learned from our datasets. Section 4 describes how our experiments were conducted and introduces the used datasets. Section 5 presents the results obtained from our experiments. Section 6 addresses the reproducibility of our experiments and our AI usage. Section 7 discusses these results and answers our research questions. Finally, section 8 gives a summary of our obtained results and discusses any future work.

2 Preliminaries

The following sections introduce the background underlying this work. The key concepts covered are extended automata, state splitting and merging, the FlexFringe framework, and information criteria.

2.1 Extended Finite State Machine (EFSM)

An EFSM extends the definition of a normal automaton by extending the transition function with guards and attributes. Guards are conditions that decide whether a transition can be taken. This allows the automaton to define a transition based not only on the current state and symbol but also on the values of the attributes.

Figure 1 illustrates an extended DFA for the alphabet $\Sigma = \{a\}$. The automaton contains two states, where q_0 is the initial state and q_1 is the accepting state. The transition from q_0 to q_1 can only take place when the next symbol is a and $x < 1.3$. When $x \geq 1.3$, the automaton transitions back to q_0. The transition from q_1 to q_0 happens when the next symbol is a for any x. This example demonstrates how guards allow for more control over the automaton.

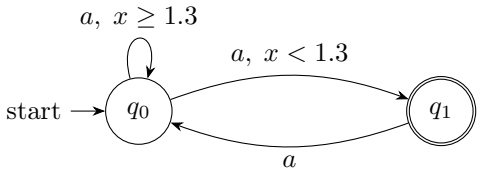


Figure 1: Example of an extended DFA with alphabet $\Sigma = \{a\}$ and guards on the variable x.

2.2 Extended regression automata (ERA)

An ERA is a special type of extended automaton that has the properties of EFSMs, such as guards and attributes. An ERA differs from most automata in that it does not have accepting or rejecting states, but instead uses a function $P(q)$ that assigns a numerical value to each state q, representing its prediction. This allows automata to be used for regression tasks.

Figure 2 shows an example of a regression automaton where the values in each node represent the predicted value. This means that a trace of $[2, 1]$ will result in a prediction of 20 while $[2, 3]$ results in a prediction of 10.

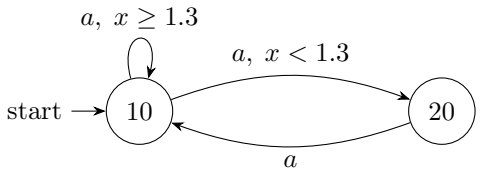
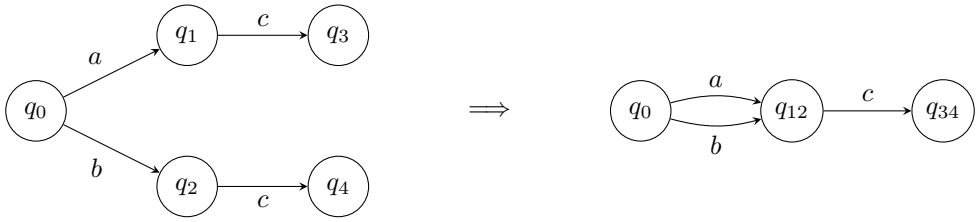


Figure 2: Example of a learned regression automaton.

2.3 State merging and splitting

Merging and splitting are the primary operations that are used when learning automata. Both operations are allowed only when they pass the consistency checks. These checks determine whether the operation sufficiently improves the model (subsection 3.3).

When two states are merged, they are combined into a single state. This is done by making the incoming transitions from the old states point to the new combined state, and all the outgoing transitions from the old states point to the old outgoing states. If the merge creates multiple transitions with the same label and guard from one state, the target states must also be merged to preserve the deterministic property. Figure 3 shows an example of a merge.



(a) Before merge

(b) After merge

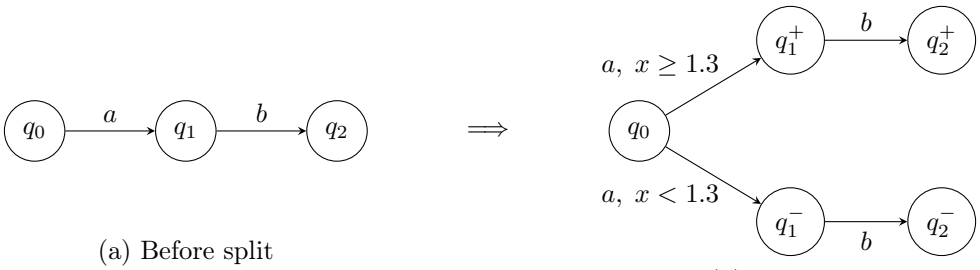
Figure 3: Example of a merge. Merging q_1 and q_2 creates one outgoing transition labelled c from the merged state and points both incoming transitions to the new state. The combination of states now receives the same prediction while previously receiving different ones.

When a state is split, two new states are added. The incoming transition from the old state is now turned into two separate transitions with a guard. This guard sends higher values to the first state and lower values to the second state. This will cascade further, just like merging. Figure 4 shows an example of a split.

2.4 FlexFringe

FlexFringe [5] is an automata learning framework that allows for learning different types of automata. It has a built-in red-blue algorithm for state merging and splitting. This algorithm starts with a prefix tree acceptor for all the traces. The root node is then marked red, and all its children are marked blue.

The algorithm then continually attempts to merge a blue node with a red node or split a blue node. Each time, it checks whether the resulting DFA is still consistent. If it is, the process continues, and if it is not, the operation is reverted. A blue node is promoted to



(a) Before split

(b) After split

Figure 4: Example of a state split. State q_1 is replaced by two states separated by a guard on attribute x. From q_0, all traces with $x > 1.3$ are pointed to q_1^+, while traces with $x \leq 1.3$ are pointed to q_1^-. This separation allows for different predictions for both q^+ and q^-.

a red node if no consistent split or merge can be found. This process is repeated until all nodes are red, at which point we have the final automaton. Any non-red state that has a transition from a red node to it is considered blue.

Figure 5 shows an example of the red-blue algorithm. Examples of possible operations are merging R_1 and B_2, or splitting B_1 into two new states.

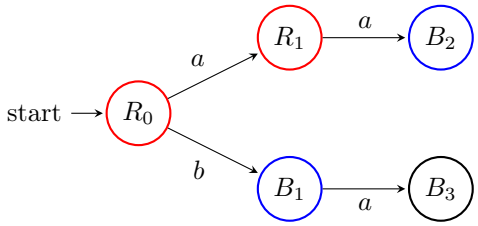


Figure 5: Example of the red-blue framework. Red states have already been processed, blue states are candidates, and black states are states that are not yet considered.

2.5 Information criteria

Information criteria are used to evaluate the trade-off between model complexity and the fit to the dataset. A better fit is often rewarded, while an increased complexity is penalized.

2.5.1 Akaike Information Criterion (AIC)

The AIC [6] is a popular method for estimating the quality of a model. If we assume least-squares fitting, the difference in AIC can be expressed in terms of the Residual Sum of Squares (RSS). The first term ($2k$) penalizes increases in complexity, while the second term rewards a reduction in RSS.

$$\Delta AIC = 2k + n \ln\left(\frac{RSS_{before}}{RSS_{after}}\right). \tag{1}$$

where k represents the parameter for the model complexity, and n is the number of data points.

2.5.2 Bayesian Information Criterion (BIC)

The BIC [7] is closely related to AIC. The difference lies in the first part of the function, since BIC uses $(k \ln(n))$ while AIC uses $(2k)$. This penalty grows with the number of data points. This means that larger datasets are penalized for complexity. This often results in smaller models when compared to AIC.

$$\Delta BIC = k \ln(n) + n \ln\left(\frac{RSS_{before}}{RSS_{after}}\right). \tag{2}$$

where k represents the parameter for the model complexity, and n is the number of data points.

3 Methods

This section describes the process of turning the data into an extended automaton. This includes the data processing, the learning process, and the consistency check.

3.1 Preprocessing of data

The sequential dataset is converted into traces for FlexFringe. This is done by turning the dataset into a set of sliding windows of a fixed size. Each sliding window is given an ID, which is the trace ID. Each value in the window is set as an attribute of the same symbol. This symbol is used for the entire dataset. The resulting traces can then be used to learn the automata.

An example of a trace is shown in Listing 1. Each row represents a symbol with its corresponding data. The column `ff_id` identifies the trace to which the transition belongs, while `ff_symb` specifies the symbol. In this example, all transitions belong to trace 0 and use the same symbol, `a`. The column `ff_ttype` denotes the trace type and is set to 0 for all transitions. Finally, `ff_sattr:stf/value` contains the symbol attribute named `value`. The flags specify that the attribute is splittable and a target. A splittable attribute can be used in guard conditions when a state is split. A target attribute specifies the value that the model is intended to predict.

Listing 1: Example trace input

```
ff_id, ff_symb, ff_ttype, ff_sattr:st/value
0, a, 0, -0.0758
0, a, 0,  1.0929
0, a, 0, -0.0488
```

3.2 Extended automata learning

To learn the automata, we make use of an MSE-based error function. This allows us to use the RSS, which is used in our AIC/BIC consistency checks. Another advantage is that it can be computed incrementally using counts, sums, and sums of squares at each state without storing or rescanning the traces. This is essential since there are a large number of possible splits and merges.

The counts (n), sums (S), and sums of squares (Q) are used to calculate the Residual Sum of Squares (RSS) before and after each split or merge. RSS_{before} is defined as the RSS

of the two states separated, and RSS_{after} as the RSS of the combined states. This means that for splits, the RSS_{before} and RSS_{after} are used in reverse order. The sums of n, S, and Q of the left and right states are used to calculate the RSS of the combined state.

$$RSS = Q - \frac{S^2}{n} \tag{3}$$

When the automata learning process is completed, we will obtain an extended regression automaton. Each state still contains the counts, sums, and sums of squares. These are used to calculate a mean value for each state, which is considered the predicted value for that state. An example of a regression automaton can be found in subsection 2.2.

3.3 Consistency check

The consistency checks are done using information-criterion-based scores. These scores evaluate a merge or split based on the trade-off between model complexity and goodness of fit. Merges and splits are treated differently because they have opposite effects on the model structure. A merge reduces the number of states and therefore simplifies the model, while a split increases the number of states and therefore increases the model complexity.

Before a merge is evaluated with the information criterion, a local consistency check is used as an early stopping condition. This check compares the variance of the target values in the two states that are considered for merging. If the variance of one state is more than two times larger than the variance of the other state, the merge is rejected immediately, and the remaining recursive merge steps are not evaluated. This prevents states with different output variance from being merged. This local variance check is only used for merges. Splits are evaluated directly with the split consistency score.

In our work, we make use of the AIC and the BIC, which were introduced in subsection 2.5. These functions make use of the RSS both before and after each operation, which we calculated earlier in the learning process. We also use the number of merges as the complexity parameter. When scoring a merge, the complexity term is used as a reward because the merge simplifies the model. When scoring a split, it is used as a penalty.

The operation is considered consistent when the calculated value is above a parameter that we call the `confidence_bound`. To show this, we consider a simple example where two states are merged. The RSS before the merge is $RSS_{before} = 10.0$, and after the merge, it becomes $RSS_{after} = 8.0$. Furthermore, we have only performed one merge so far, so $k = 1$, and it is evaluated on $n = 100$ data points.

For AIC, the change would be

$$\Delta AIC = 2k + n \ln\left(\frac{RSS_{before}}{RSS_{after}}\right) = 2 + 100 \ln\left(\frac{10}{8}\right) \approx 24.31. \tag{4}$$

Since $\Delta AIC \approx 24.31$, the merge is accepted when the `confidence_bound` is lower than this value.

The main difference between AIC and BIC is in the complexity penalties. AIC uses a fixed complexity term of $2k$, while BIC uses $k \ln n$. While the coefficient in the AIC penalty is fixed at 2, the BIC penalty increases with the number of data points n. In the case of merges, this means that BIC gives a stronger reward to model simplification than AIC for sufficiently large datasets. As a result, BIC tends to favor simpler models and may accept state merges that slightly worsen the fit to the data if the reduction in complexity is large enough. In contrast, for splits, the larger BIC complexity term makes it harder for a split

to be accepted, because the improvement in fit must compensate for the increased model complexity. This makes BIC generally more conservative with respect to model complexity and less prone to overfitting than AIC.

4 Experimental setup

4.1 Methods for comparison

In our work, we compared the performance of extended automata with the following methods:

- **Persistence** is a simple baseline method that predicts the next value by using the current value. This is often used as a baseline method for sequential regression.
- **Regression tree (RT)** is a method that works like a decision tree but predicts a continuous value. We use the implementation from sklearn [8], which uses an optimized version of the CART algorithm [9] to build the tree.
- **Random Forest Regressor (RFR)** is a collection of RTs. The implementation in sklearn is based on [10]. For regression tasks, the final result is obtained using the average of the predicted values rather than using the majority of votes, which is used for classification.

4.2 Datasets

In this section, we introduce the datasets we will use in our experiments. Each subsection includes a brief description of how we obtained the dataset and what the data represent.

The datasets are chosen to test the method on increasingly difficult problems. The noisy sine dataset is a simple controlled example where loops should be useful. The Mackey-Glass dataset is a more complex deterministic sequence with periodic behaviour. The hourly electricity consumption dataset is a noisy real-world dataset, which tests whether the method also works outside synthetic examples.

To ensure a fair comparison across methods, each dataset is partitioned into three subsets: training (70%), validation (15%), and test (15%). The training dataset is used to train the RT, RFR, and automaton. We then use the validation set to tune any hyperparameters. Finally, we use the test set to evaluate our results. These splits are done in chronological order to preserve sequential dependencies and avoid training on future observations.

4.2.1 Noisy sine

Noisy sine is a simple dataset based on the sine function. We always start the traces from 0, but we alter whether the next value is 1 or -1, and the steps are always of size $\frac{1}{2}\pi$. It also includes uniformly distributed noise with $U(-0.1, 0.1)$ on top of the value produced by the sine function.

4.2.2 Mackey-Glass

The Mackey-Glass [11] is a function that is used in mathematical biology. It shows periodic behaviour just like the sine function, but it is a bit more complex. The values were produced using the following function:

$$x_{t+1} = x_t + dt\left(\frac{\beta_0 x_{t-\tau}}{1 + x_{t-\tau}^n} - \gamma x_t\right). \tag{5}$$

where $x_{t-\tau}$ is the delayed value of the system, x_t is the current state, and x_{t+1} is the next state. The parameter τ controls which delayed value is used in the calculation, while dt is the time step used to generate the values. The parameters β_0 and γ control the dynamics of the system but are not the focus of our research. These are set to $\beta_0 = 0.1$ and $\gamma = 0.2$. For all our testing, we use $dt = 2$ and $\tau = 12$. We also discard the first 400 points, as the function still needs to warm up, which is standard practice.

4.2.3 Hourly electricity consumption

The hourly electricity consumption dataset [12] contains the electricity consumption of 321 clients in kilowatt-hours (kWh). It was originally recorded in 15-minute increments, but we are making use of the aggregated version, which shows the average over an hour. The data originates from 2011 to 2014. During our experiments, we make use of the data from the client named "T2".

4.3 Evaluation metrics

Evaluation metrics are used to evaluate the effectiveness of the resulting models. Let $x = [x_1, x_2, \ldots, x_n]$ denote the predicted values and $y = [y_1, y_2, \ldots, y_n]$ the true values. We make use of the following metrics:

- Mean Absolute Error (MAE):

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{6}$$

- Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y_i - x_i)^2}{n}} \tag{7}$$

- Mean Absolute Percentage Error (MAPE):

$$MAPE = 100 \times \frac{\sum_{i=1}^{n} \left|\frac{y_i - x_i}{y_i}\right|}{n} \tag{8}$$

4.4 Confidence bound selection

The selection of the confidence bound is a hard problem since there is no commonly used strategy. We validate our choice using a combination of two different methods.

The first method is based on the number of nodes and the shape of the automaton. When we look at the resulting DFA and see that it has too many states or states we do not expect, we increase the confidence bound. When we see too few, or we miss some properties, we decrease it.

The other method uses the learned automaton's accuracy on both the training and validation sets. We use them to check for large differences, which indicate overfitting. We do this before checking the accuracy of the test set, to make sure the final evaluation is unbiased.

4.5 Hardware

All of our tests were run on an HP Zbook Power G10 laptop. This system uses an Intel 13th-generation i7-13700H and 16GB of RAM. It runs the Linux operating system and uses the 7.0.9-arch2-1 kernel.

5 Results

This section shows the results obtained from our experiments on the noisy sine, Mackey-Glass, and hourly electricity consumption datasets. The results are analysed to evaluate the effectiveness of the models.

5.1 Case 1: Noisy sine

Looking at the results of the noisy sine experiment in Table 1, all models except the persistence baseline achieve MAE values close to 0.05. This is expected because the noise added to the signal is uniformly distributed within ± 0.1, resulting in an expected absolute error of approximately 0.05. In contrast, persistence performs significantly worse than the others. It shows a mean absolute error of 1.0103, which is also expected since always guessing the previous value will mean you're always off by approximately 1.

Metric	Persistence	Regression tree	Random forest	BIC	AIC
MAE	1.0103	0.0495	0.0489	**0.0460**	0.0475
RMSE	1.0131	0.0572	0.0554	**0.0535**	0.0551
MAPE (%)	99.8041	4.8908	4.8172	**4.5205**	4.7968

Table 1: Next value prediction model performance comparison for noisy sine (lower is better).

Our results also show that MSE with BIC produces the lowest value for the MAE, RMSE, and MAPE. However, the differences between BIC, regression tree, random forest, and AIC are very small. We can conclude from this that all models except for persistence can learn the dynamics of this simple system.

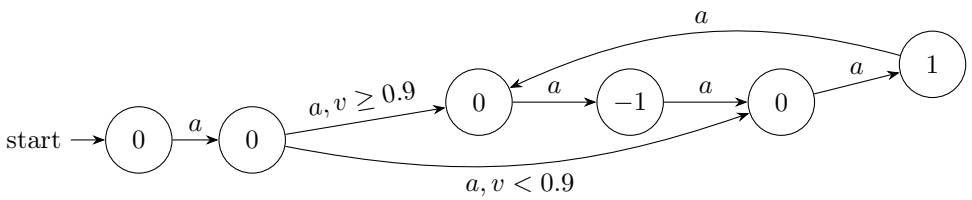


Figure 6: An illustration of the regression automaton produced by BIC with values rounded to a precision of 0.1. The value in each node represents the predicted value.

Figure 6 shows the automaton produced using BIC. Each state contains the mean predicted next value for that state. The initial state has a predicted value of 0 because every trace starts at 0. The following state also has a mean prediction close to 0 because the automaton does not yet know whether the signal is increasing or decreasing. As a result, the mean of both possibilities gives an average prediction of approximately 0. After this

point, there is a split at 0.9, since the automaton now knows where it is in the cycle. These transitions point to the appropriate state inside a loop that cycles between all the values.

Compared to AIC, the automaton from BIC is more compact, containing only 6 states instead of 8 and fewer transitions. This is due to the stronger complexity penalty of BIC. This means BIC is more likely to merge similar states.

We also found in our testing that the minimum window size is 3, for which both automata still contain a cycle, although their configurations are different. The automaton from BIC still has 6 states but in a slightly different configuration, while performing the same as the automaton shown in Figure 6. In contrast, the automaton from AIC grows in size to 15 states. We cannot reduce the window size any further. With a window size of 2, the predicted value is always 0, while with a window size of 1, the model is effectively making a random guess.

5.2 Case 2: Mackey-Glass

Looking at the results for a window size of 4 from the Mackey-Glass experiment shown in Table 2, we see that Random Forest performs the best. The regression tree follows, while AIC and BIC perform worse, with AIC slightly better than BIC. When we look at the sizes of each automaton, we see a bigger difference, since BIC has a size of 199 while AIC has a size of 301.

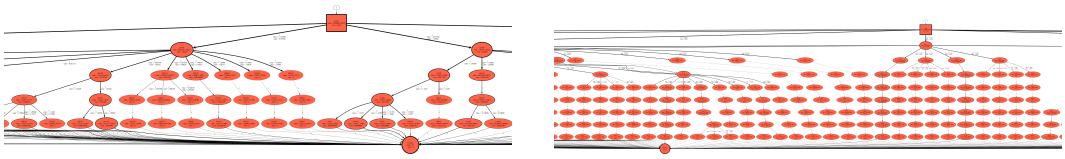
Metric	Persistence	Regression tree	Random forest	BIC	AIC
Window size = 4					
MAE	0.0647	0.0054	**0.0050**	0.0100	0.0082
RMSE	0.0766	0.0084	**0.0068**	0.0152	0.0109
MAPE (%)	7.6123	0.6413	**0.5961**	1.1001	0.9340
Window size = 8					
MAE	0.0616	0.0060	**0.0029**	0.0056	0.0055
RMSE	0.0736	0.0132	**0.0042**	0.0111	0.0126
MAPE (%)	7.1554	0.6463	**0.3440**	0.6354	0.6066
Window size = 48					
MAE	0.065	0.0021	**0.0012**	0.0175	0.0223
RMSE	0.0769	0.0029	**0.0016**	0.0238	0.0311
MAPE (%)	7.601	0.2343	**0.134**	2.0156	2.4803

Table 2: Next value prediction model performance comparison for different window sizes on the Mackey-Glass dataset (lower is better).

When we look at the results for a window size of 8, the results change. Random Forest still performs the best, but now BIC and AIC perform better than the regression tree. However, when we look at the sizes of the automata, we see something concerning. The automaton learned by BIC now has a size of 1312, while the one from AIC has a size of 1528.

Figure 7 shows the automata produced by BIC for both window sizes. Both automata resemble trees whose depth equals the window size, with no loops in either automaton, but many states. This suggests that the learning process mainly performs splits and rarely performs successful merges, since loops can only appear when states are merged.

There are two possible explanations for why these structures resemble trees. The first is that the precondition for merging may be too strict, causing useful merges to be rejected.



(a) Window size = 4 (b) Window size = 8

Figure 7: Subsections of regression automata produced by BIC for two different window sizes on the Mackey-Glass dataset.

The second is a bias in the scoring function towards splitting, since splits tend to reduce the RSS by allowing states to specialize. We first investigate the effect of the precondition.

We tested the effect of the local consistency condition by first increasing the variance ratio used in the check. However, increasing this ratio did not noticeably affect the learned models. Therefore, we also performed an experiment in which the local consistency condition was skipped entirely. The results of this experiment are shown in Table 3.

The results show that all methods outperform the persistence baseline. The regression tree and random forest have lower error than both AIC and BIC. However, the most important change is observed for BIC. Without the precondition, BIC learns a model with loops and a size of only 43 states. In contrast, AIC still does not learn any loops and produces a much larger model with 2256 states. The BIC run also finished in just 26 seconds, while AIC took 108 seconds.

This makes BIC the smallest overall in this experiment. The regression tree contains 256 nodes, while the random forest contains more than 15,000 nodes. These results suggest that the local consistency condition has a strong influence on the structure learned by BIC. However, removing this precondition comes at the cost of accuracy, since the error roughly doubles when the condition is removed.

Metric	Persistence	Regression tree	Random forest	BIC	AIC
MAE	0.065	0.0021	**0.0012**	0.0476	0.0207
RMSE	0.0769	0.0029	**0.0016**	0.0673	0.0299
MAPE (%)	7.601	0.2343	**0.134**	5.3093	2.2801

Table 3: Next value prediction model performance comparison for a window size of 48 without precondition (lower is better).

The second possible reason for these structures is a bias towards splitting. A split almost always improves the RSS, because it allows states to specialize and fit the traces more closely. A merge has the opposite effect, since it combines states that may have different predicted values and therefore often increases the RSS. The complexity term in both information criteria is intended to prevent this bias by penalizing additional complexity. However, the logs from FlexFringe show that this penalty is not strong enough to overcome this. They show that merge scores are consistently lower than split scores.

As a result, the model continues to prefer splits until the traces are represented with very low error. At that point, the RSS approaches zero. This makes later merges difficult because any merge that increases the RSS receives a very low score. This also means further merging becomes unlikely, and the automaton remains large and tree-like.

This also explains the unexpected result for the Mackey-Glass dataset. We expected

that this dataset would be simpler to learn since it does not contain any noise. This was expected to make the Mackey-Glass pattern easier to learn. However, the results show that the learned automata are still strongly affected by the interaction between the precondition, the scoring function, and the preference for splitting.

5.3 Case 3: Hourly electricity consumption

The hourly electricity consumption dataset is our first real-world dataset. In the last experiment for Mackey-Glass, we found that the learned automata look like trees. We also found two possible reasons for this behaviour: the precondition for merging and the bias towards splitting. In this experiment, we look at the effect of the precondition on a dataset that does contain noise.

Looking at the results shown in Table 4, we see that both BIC and AIC perform the worst among all the methods, including persistence. None of the automata learned with or without the precondition contain loops. This is because we had to increase the `confidence_bound` to a much higher value to prevent overfitting. Since the previous experiment showed that there is already a bias towards splitting, this higher threshold makes merges even less likely. As a result, the automata remain tree-like, even after removing the precondition. This shows that the main problem is related to our information criteria, or more specifically, the complexity parameter.

Metric	Persistence	Regression tree	Random forest	BIC	AIC
With precondition					
MAE	7.0340	6.6950	**4.5062**	7.5271	7.824
RMSE	10.1880	10.7446	**7.0153**	10.7095	12.6665
MAPE (%)	6.0293	5.6540	**3.7734**	6.3441	6.5616
Without precondition					
MAE	7.0340	6.6950	**4.5062**	8.251	7.824
RMSE	10.1880	10.7446	**7.0153**	11.6138	12.6665
MAPE (%)	6.0293	5.6540	**3.7734**	7.0606	6.5616

Table 4: Comparison of accuracy for automata learned with and without the precondition for the hourly electricity consumption experiment (lower is better).

For AIC, the error is the same with and without the precondition. The `confidence_bound` also stayed the same for both tests. This suggests that the precondition is not a limiting factor for AIC in this experiment. A possible explanation is that the `confidence_bound` is set so high that the merges rejected by the precondition would not have been accepted by the scoring function anyway.

For BIC, removing the precondition has a clearer effect. The size of the automaton decreases from 797 to 506 states, but the accuracy also decreases. This shows that the precondition does influence the structure learned by BIC. This is similar to what we found in the Mackey-Glass experiment, where removing the precondition allowed BIC to produce a smaller model, but at the cost of prediction accuracy.

6 Responsible research

6.1 Reproducibility

This paper includes links to the two repositories containing all the code used in our experiments. These include Jupyter notebooks for our tests[1] as well as a modified version of FlexFringe[2]. The modifications add support for BIC and AIC and allow for exporting the automata for use in the experiments. Our experiments use a fixed seed of 42 to allow for reproducibility. Small numerical differences may still occur due to numerical instability. All the parameter settings are included in the code and also displayed in the Appendix of this paper.

6.2 AI usage

During our research, we used AI in the following ways:

- To assist with formatting tables and figures in LaTeX. The resulting tables and figures were always checked by the author to prevent hallucinated results.
- To assist with writing code when no solution could be found after searching the internet, or when an issue was encountered without a clear solution.
- To detect any grammatical errors that were not found by the spelling checker.

All our research activities, including literature review, methodology, and analysis, were conducted by the author. All AI-generated contributions were thoroughly checked.

7 Discussion

The noisy sine dataset shows that extended automata can learn simple cyclic behaviour. The resulting automaton clearly contains a loop, and when we follow this loop using the data, we find that it matches the underlying periodic structure well. This is expected, since the noisy sine dataset has a simple repeating pattern and only a small amount of noise.

For both the Mackey-Glass and hourly electricity consumption datasets, we found the opposite behaviour. For these datasets, the resulting automata mostly resembled trees without any loops. As a result, the automata became very large when the window size increased. This limits the main advantage of using extended automata, because without loops, the learned models behave more like regression trees.

We investigated two possible causes for this behaviour. The first is the precondition for merging. This precondition rejects merges when the ratio between the variances of two states is greater than 2. Removing this condition allows more states to be merged. For the Mackey-Glass dataset, this made a large difference for BIC, while AIC changed little. This suggests that, for BIC, many merges that score well according to the information criterion are blocked by the precondition. This can be explained by the stronger complexity term in BIC, together with the fact that BIC often uses a lower `confidence_bound`, which makes merges more likely to be accepted. However, the electricity consumption dataset shows that removing the precondition is not enough when the dataset contains more noise. It also

[1] https://github.com/jobvonck/paper_code
[2] https://github.com/jobvonck/Flexfringe

decreases the accuracy of BIC, because states with different output variances can then be merged. Therefore, removing the precondition is not a good general solution.

The second possible cause is the preference for splitting. Splits almost always reduce the prediction error, because they allow states to specialize in different parts of the data. Merges have the opposite effect, since they combine states that may have different predictions. This can increase the prediction error, even if the merge would make the automaton smaller and introduce useful loops. The complexity terms in AIC and BIC were intended to compensate for this, but they were not strong enough in our experiments. As a result, the learning process often prefers splitting over merging. This suggests that the main limitation is not the use of extended automata itself, but the way model complexity is currently measured during learning.

This also explains why the automata become tree-like for the more complex datasets. When the algorithm keeps accepting splits, the model can fit the training traces more closely. However, this also reduces the opportunity for later merges. Once the model has already split the data into many specialized states, merging them again becomes less attractive because it increases the RSS. Therefore, the automaton grows larger and remains closer to a tree structure.

8 Conclusions and future work

This paper investigates whether extended automata learned by an MSE-based approach can improve sequential regression by using loops to model cyclic patterns. Two variants of the learning method were tested: one used AIC to score merges and splits, and the other used BIC. We compared them against persistence, regression tree, and random forest baselines on three datasets: a noisy sine dataset, the Mackey-Glass dataset, and the hourly electricity consumption dataset.

The main conclusion is that the learned extended automata did not outperform the regression tree and random forest baselines overall. We found that the extended automata achieved better performance than the regression tree and random forest for the noisy sine dataset and successfully learned the simple cyclic structure. However, for the two more complex datasets, Mackey-Glass and hourly electricity consumption, the automata performed worse and mostly resembled trees instead of using loops.

This tree-like structure is the most important explanation for the negative result. It is caused by a bias towards splitting in the scoring function, where splits almost always improve the RSS while merges often worsen it. The complexity parameter, which is based on the number of merges, is not able to overcome this bias. This severely limits the benefits of using extended automata, since without loops, the automata behave more like a limited version of a regression tree.

We also found that BIC consistently produced smaller automata than AIC across all three datasets and ran faster when using large windows. In terms of accuracy, the results are more mixed. On the noisy sine dataset, both criteria perform similarly. On the more complex datasets, AIC occasionally achieved lower error than BIC, particularly for larger window sizes. We can therefore conclude that BIC is preferable when model compactness and learning speed are important, while AIC may be worth considering when peak accuracy is the priority.

Future work includes investigating other learning methods such as RTI [13]. Another possible improvement is to use a different complexity parameter, such as the total automaton size.

A Input parameters

A.1 Noisy sine

Parameter	Value
windowSize	7
numWindows	200
Regression Tree max_depth	3
Random Forest max_depth	4
Random Forest n_estimators	100
BIC confidence_bound	1
AIC confidence_bound	1

Table 5: Hyperparameters for the noisy sine experiment.

A.2 Mackey-Glass

Parameter	Window size 4	Window size 8	Window size 48
windowSize	4	8	48
numWindows	500	500	1000
Regression Tree max_depth	7	8	9
Random Forest max_depth	7	7	9
Random Forest n_estimators	10	20	30
BIC confidence_bound	15	15	15
AIC confidence_bound	20	20	20

Table 6: Mackey-Glass hyperparameters for different window sizes.

Parameter	Value
windowSize	48
numWindows	1000
Regression Tree max_depth	8
Random Forest max_depth	9
Random Forest n_estimators	30
BIC confidence_bound	15
AIC confidence_bound	10

Table 7: Mackey-Glass hyperparameters for the case without the precondition.

A.3 Hourly electricity consumption

Parameter	Without precondition	With precondition
windowSize	24	24
numWindows	1000	1000
Regression Tree max_depth	8	8
Random Forest max_depth	8	8
Random Forest n_estimators	100	100
BIC confidence_bound	125	125
AIC confidence_bound	200	200

Table 8: Hyperparameters for hourly electricity consumption.

B Datasets

B.1 Mackey-Glass

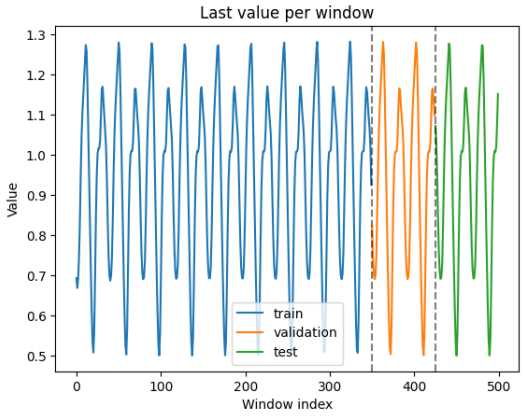


Figure 8: All data points for the Mackey-Glass function divided into their respective sets.

B.2 Hourly Electricity Consumption

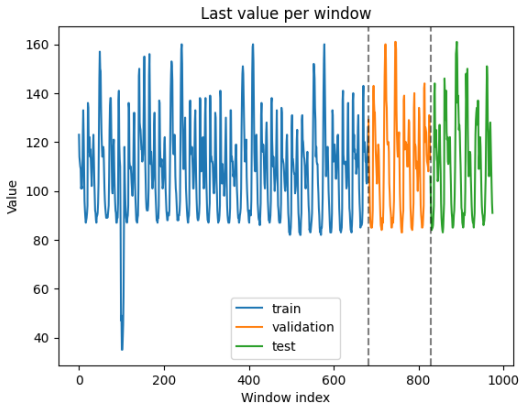


Figure 9: All data points for the energy consumption dataset divided into their respective sets.

References

[1] C. B. Pop, V. R. Chifu, C. Cordea, E. S. Chifu, and O. Barsan, "Forecasting the Short-Term Energy Consumption Using Random Forests and Gradient Boosting", 2022. DOI: 10.48550/ARXIV.2207.11952.

[2] Y. Liu and H. Wu, "Prediction of Road Traffic Congestion Based on Random Forest", in *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, Hangzhou: IEEE, 2017, pp. 361–364. DOI: 10.1109/ISCID.2017.216.

[3] N. Walkinshaw, R. Taylor, and J. Derrick, "Inferring extended finite state machine models from software executions", *Empirical Software Engineering*, vol. 21, no. 3, pp. 811–853, 2016. DOI: 10.1007/s10664-015-9367-7.

[4] Q. Lin, C. Hammerschmidt, G. Pellegrino, and S. Verwer, "Short-term Time Series Forecasting with Regression Automata", 2016. DOI: 10.13140/RG.2.2.32111.84646.

[5] S. Verwer and C. Hammerschmidt, "FlexFringe: Modeling Software Behavior by Learning Probabilistic Automata", *Logical Methods in Computer Science*, vol. Volume 21, Issue 3, p. 9295, 2025. DOI: 10.46298/lmcs-21(3:31)2025.

[6] H. Akaike, "A new look at the statistical model identification", *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, 1974. DOI: 10.1109/TAC.1974.1100705.

[7] G. Schwarz, "Estimating the Dimension of a Model", *The Annals of Statistics*, vol. 6, no. 2, 1978. DOI: 10.1214/aos/1176344136.

[8] F. Pedregosa et al., "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[9] L. Breiman, Ed., *Classification and regression trees* (The Wadsworth statistics/probability series). Belmont, Calif: Wadsworth International Group, 1984.

[10] L. Breiman, "Random Forests", *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. DOI: 10.1023/A:1010933404324.

[11] M. C. Mackey and L. Glass, "Oscillation and Chaos in Physiological Control Systems", *Science*, vol. 197, no. 4300, pp. 287–289, 1977. DOI: 10.1126/science.267326.

[12] A. Trindade, *ElectricityLoadDiagrams20112014*, 2015. DOI: 10.24432/C58C86.

[13] S. Verwer, M. De Weerdt, and C. Witteveen, "A Likelihood-Ratio Test for Identifying Probabilistic Deterministic Real-Time Automata from Positive Data", in *Grammatical Inference: Theoretical Results and Applications*, D. Hutchison et al., Eds., vol. 6339, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 203–216. DOI: 10.1007/978-3-642-15488-1_17.