NATIONAL RESEARCH UNIVERSITY
HIGHER SCHOOL OF ECONOMICS

Faculty of Computer Science
Bachelor's Programme "Data Science and Business Analytics"

UDC 004.02

Research Project Report on the Topic:

Chaotic Time Series Forecasting

Submitted by the Student:

group #БПАД222, 3rd year of study — Fatianov Vladimir Andreevich

Approved by the Project Supervisor:

Tomashchuk Korney Kirillovich
Research Fellow
Faculty of Computer Science, HSE University

Moscow 2025

Contents

Annotation

The aim of this project is to create machine learning model, capable of predicting chaotic time series many steps ahead. It is planned to use generalised Z-vectors and decision trees to predict points and suppress exponential growth of the error. Additionally, a classifier will be trained to determine whether a given point is predictable or not.

Аннотация

Целью данного проекта является написание модели машинного обучения, способной предсказывать хатоические ряды на много шагов вперед. Для предсказания точек и подавления экспоненциального роста ошибки планируется использовать обобщенные Z-вектора и решающие деревья. Также планирется обучить классификатор, который будет определять является ли точка предсказуемой или нет.

Keywords

Machine Learning, Time Series, Decision Tree, Multi Step Prediction

1 Introduction

Huge amount of data, for instance in healthcare or in energy industry, has a form of chaotic time series. Prediction of such data, especially on multiple steps ahead, is quite complicated, thereby creation of algorithms, which would be able to accurately make predictions on multiple steps without exponential growth of error is crucial. For example, in medicine, it will be possible to predict disease attacks (such as epilepsy or heart failure) several hours before they occur, while in the energy sector, accidents can be prevented by forecasting peak loads on the grid in advance.

Chaotic time series is a series of data generated by a deterministic nonlinear dynamic system, which has properties, which are specific to chaotic processes. Despite series looking completely random, it has hidden deterministic structure, which is defined by the equations of itself. As could be seen on Figure 1.1, some parts of the series are repeated. Hence, compared to stochastic time series, in which future values are completely independent from previous values and are fully random, chaotic series heavily relies on the initial conditions and inner laws. Moreover, chaotic series is bounded and is oscillating within specific range, and not diverging to infinity, the-

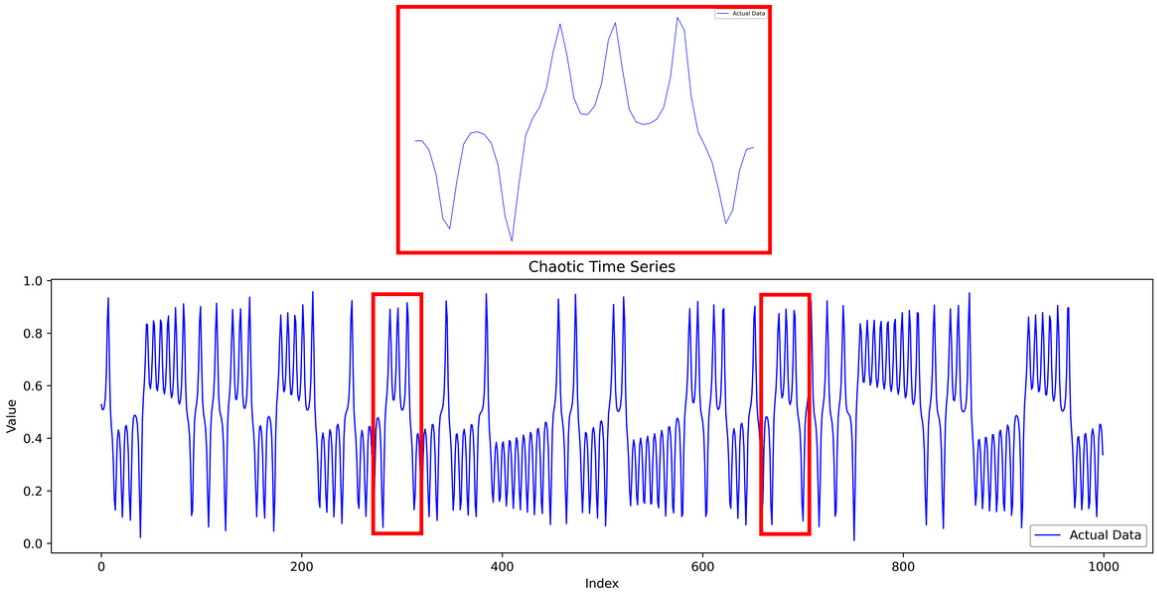


Figure 1.1: Example of recurring motifs

oretically making short term predictions possible. Nevertheless, due to the exponential sensitivity to the initial conditions, long-term prediction of following series is by now is impossible.

Main goal of project - development of prediction method, that avoids exponential growth of an error due to the usage of generalised Z-vectors. Unlike conventional methods, like LSTM [3, 7], Z-vectors are built based on non-adjacent points of the time series, which allows for a better consideration of its hidden structure. Herewith, ensemble of decision trees would be used to make a prediction.

2 Literature review. Description and characterisation of relevant works

There are plenty of methods used to predict chaotic series—for example, the classic delay-coordinate embedding and local prediction method of Farmer Sidorowich (1987) [4], but all of them are unable to resolve the problem of exponentially growing error [5]. Several studies have attempted to tackle this issue in their proposed algorithms but ultimately struggled to overcome it [6]. However, there is an algorithm which stands out against the background of classical algorithms, it was published by Gromov and Baranov in 2023 [2][8]. Currently, it is one of the most promising algorithms, since unlike traditional methods, it considers local data structures and dynamically adapts to their complexity. However, the very recent approach by Giammarese et al. (2024) [1] also shows considerable promise, although its novelty means it has yet to be extensively validated.

As mentioned earlier, a key element of predictive clustering is usage of Z-vectors - special representations of time series, built on the basis of non-adjacent points. This approach allows to identify recurring motifs and use their features for prediction. Additionally, it enables the algorithm to detect deep patterns in data, while ingnoring the white noise.

Another special features of this algorithm - ability to work based on local principle, while other algorithms are building unified global prediction. Predictive clustering method works as following: First, chaotic time series is divided on clustered patterns. Then, during the prediction sub-series, which mostly resembles our target vector, is selected. If selected sub-series has small absolute difference compared to target vector, its continuation is used for forecast. Otherwise, point is classified as unpredictable and skipped.

This method minimises the number of unpredictable points, ensuring that prediction is made only when possible, thus preventing the model from being clogged with false points. Since the model is not clogged with the false points and predictions are only made when necessary - the mean error is significantly reduced. The algorithm also performed well in tests on both synthetic and real-world data, Lorenz series and data regarding electrical consumption in Germany, respectively, showing following results:

- Error on predictable points remained stable even when the forecast horizon was increased.
- Algorithm has been able to successfully skip unpredictable points.
- Increase in data volume led to decrease in the number of unpredictable points, confirming the effectivness of clustering.

3 Original algorithm

First and foremost, original algorithm, alongside with some functions/methods (such as fit), which are shared by both original solution and my algorithm, had been implemented.

The original forecasting algorithm is based on the concept of repetition of patterns with different lags in chaotic time-series. It searches for patterns whose Euclidian distance is close to the current pattern and then averages their future value.

```
function fit(X, l, k):
    N ← length(X)
    self.X ← copy of X

    # Generate every pattern α = [1, a1, …, al], each ai ∈ [1..k]
    self.alphas ← []
    for each (a1,…,al) in {1..k}^l:
        α ← [1, a1, …, al]
        append α to self.alphas

    # For each pattern, build its history matrix
    for each α in self.alphas:
        offs[0] ← α[0]
        for i from 1 to l:
            offs[i] ← offs[i-1] + α[i]
        max_start ← N − offs[l]
        rows ← []
        for start in 0 .. max_start-1:
            row_indices ← [start + offs[j] for j in 0..l]
            append row_indices to rows
        Z_matrix ← X[rows]           # shape: (max_start, l+1)
        self.Z[tuple(α)] ← Z_matrix

    return self
```

Figure 3.1: Example of fit method

3.1 Fit

3.1.1 Input and storage

The input is a one dimensional NumPy vector X of length N — initial chaotic time-series. Later copy of X and value N are saved. X is going to be used for matrix creation and N determines length of series.

3.1.2 Generation of all Lag patterns α

Next, all patterns, which are responsible for lag window creation, are built. Pattern has a following

$$\alpha = [1, a_1, a_2, \ldots, a_l],$$

where length of α is equal to $l + 1$. First element is always fixed at 1 - ensuring that every lagged window includes the current observation, while remaining l elements take integer values in the range of $[1..k]$. In total, there are k^l distinct patterns.

3.1.3 Building Z-matrix for each α

Once all α patterns have been generated, a matrix of every window matching each pattern α is constructed. First we compute cumulative offsets:

$$\begin{aligned}
\text{offs}[0] &= \alpha[0], \\
\text{offs}[1] &= \alpha[0] + \alpha[1], \\
&\vdots \\
\text{offs}[l] &= \alpha[0] + \alpha[1] + \cdots + \alpha[l].
\end{aligned}$$

Where maximum start index is equal to $N - \text{offs}[l]$. For each start from 0 to $N - \text{offs}[l] - 1$, following row of indexes is formed -

$$[\text{start} + \text{offs}[0], \ldots, \text{start} + \text{offs}[l]]$$

and corresponding values are extracted from X. The resulting matrix is stored in a dictionary under key α.

3.2 Predict

After creation of each α - matrix pair in `fit` method, goal of the algorithm yields to the step-by-step prediction of time-series's values, which is accomplished by searching for the most similar vectors from the past and averaging theirs' subsequent values. Thus, prediction is based not on global statistical model, but on locally recurring patterns in the chaotic time-series.

```
# target_calculation forms target vector from last values of X
function target_calculation(alphas, last_index, X):
    target_values ← {}
    for alpha in alphas:
        offsets ← reverse(cumsum(reverse(alpha[1…end])))
        idxs ← last_index - offsets
        target_values[alpha] ← X[idxs]
    return target_values
# z_prediction find similar historic windows to current
function z_prediction(alphas, Z_dict, y_alphas, l, threshold):
    preds ← []
    for alpha in alphas:
        Z ← Z_dict[alpha]
        windows ← Z[:, 0…l-1]
        dists ← [norm(windows[r] - y_alphas[alpha]) for r in rows]
        # выбираем строки ближе порога
        valid ← [r for r in rows if dists[r] < threshold]
        next_vals ← [Z[r, l] for r in valid]
        if not empty(next_vals):
            preds.append(mean(next_vals))
    return mean(preds) if not empty(preds) else NaN
# predictis used for iterative prediction
function predict(X_init, alphas, Z_dict, l, threshold, steps):
    X ← copy(X_init)
    predictions ← []
    for i in 1…steps:
        y_alphas ← target_calculation(alphas, len(X), X)
        p ← z_prediction(alphas, Z_dict, y_alphas, l, threshold)
        X.append(p)
        predictions.append(p)
        print("Step", i, "/", steps, "→", p)
    return predictions
```

Figure 3.2: Example of predict method

Target calculation function is responsible for creation of the target vector - vector in which the last element corresponds to the value to be predicted.

The target vector is constructed by extracting a subsequence from the X. For each given alpha sequence the function calculates relevant indices within the data and forms a subsequence, with last element being "absent".

Algorithm removes the leading 1, reverses the remaining elements, computes their cumulative sum, and then reverses the result again to obtain offsets. These offsets are subtracted from

the index of the end of the series to determine the indices of the last l observations. The corresponding values are extracted from X and stored in the dictionary $y\_alphas[\alpha]$. These vectors represent the current windows and used to query the historical library.

3.2.1 Matching Historical Windows and Averaging

For each pattern α, the matrix $Z[\alpha] \in \mathbb{R}^{M\times(l+1)}$ contains all past windows of length l together with the value that immediately followed each window. The algorithm extracts the first l columns of $Z[\alpha]$ - the past windows without their successors and computes Euclidean distances between each historical window and the target vector y_α. Rows whose distance falls below the threshold are selected, and their $(l+1)$-th column entries, the values that historically followed similar windows, are collected. The average of these successor values is a candidate prediction p_α for that pattern. Patterns without any sufficiently close windows are not taken into consideration. All candidate predictions $\{p_\alpha\}$ are then averaged to produce the final forecast p.

3.2.2 Iterative Accumulation of Predictions

This value p is appended to the end of the series X so that subsequent target vectors include it. The process repeats for the specified number of steps, generating a sequential list of predictions. In this way, the algorithm advances through time by continually comparing the latest windows to historical ones and constructing forecasts based on recurring local patterns.

4 Modifications of algorithm

Nevertheless, following algorithm has drawbacks - error accumulation without safeguard, uniform weighting of matches ans high sensitivity to outliers.

4.1 Vanilla Forest

In the proposed modification, decision trees are trained separately for each lag-pattern α, instead of simply averaging all future values of similar windows. This method helps to target following issues:

First, it replaces uniform weighting of all matches under a fixed distance threshold with adaptive, data-driven partitions: each tree learns which regions of the l-dimensional input space are most predictive, assigning implicit weights via its splits. Second, decision trees reduce sensitivity to outliers—anomalous windows are less likely to dominate the prediction because the tree isolates

and models them in separate branches. Third, the need to tune a hard distance threshold is relaxed: a single parameter such as `min_samples_leaf` suffices to control model complexity.

4.1.1 Per pattern tree training

Each `DecisionTreeRegressor` is bounded to specific template α. Hence, for every patter α we have a matrix of the form

$$Z[\alpha] \in \mathbb{R}^{M\times(l+1)}.$$

Where each tree is trained using following features and targets:

$$X_{\text{train}}(\alpha) = Z[\alpha]_{:,0:l} \in \mathbb{R}^{M\times l}, \qquad y_{\text{train}}(\alpha) = Z[\alpha]_{:,l} \in \mathbb{R}^{M}.$$

with tree having following hyperparameters `min_samples_leaf=1` and `random_state=42`. The resulting models $\{\text{tree}_\alpha\}$ are stored for all subsequent steps. In total k^l trees are trained.

4.1.2 Forecasting with Trained Trees

For each future step $i \geq 1$, the algorithm recomputes the target vectors y_α from the most recent l values of X. Next, it applies each stored tree to its corresponding y_α, producing predictions $\hat{p}_\alpha = \text{tree}_\alpha(y_\alpha)$, which are later aggregated these as mean of all predictions, ignoring any NaN results. Finally, received p is appended to X and algorithm repeats for the desired number of steps.

4.2 Combined Forest

The combined method integrates is based on combination of Vanilla Forest with reliability classifier and meta-regressor, which help to solve the issues with original algorithm's uncontrolled error accumulation, uniform weighting of all pattern-based predictions and high sensitivity to outliers. It proceeds in two phases: first, training base decision trees, reliability classifier to flag unreliable points and meta-regressor. Second - based on DecisionTrees' outputs mark point as reliable/unreliable and predict them with the usage of meta-classifier.

4.2.1 Base-Tree Training and Reliability Labeling

1 *Window extraction and target construction* proceed exactly as in the original method: for each lag-pattern α, build $Z[\alpha] \in \mathbb{R}^{(N-s)\times(l+1)}$.

2 *Train base decision trees.* For every α, fit a `DecisionTreeRegressor` on

$$X_{\text{train}}(\alpha) = Z[\alpha]_{:,0:l}, \quad y_{\text{train}}(\alpha) = Z[\alpha]_{:,l}.$$

3 *Generate reliability labels.* For the last combiner_points observations, predict each true value $X[i]$ by passing its target vector through all base trees to form $\mathbf{v}_i \in \mathbb{R}^{k^l}$. If $\left|\text{mean}(\mathbf{v}_i) - X[i]\right| >$ threshold, label the point as "unreliable" (1), otherwise "reliable" (0).

4.2.2 Combiner Regressor Training

Using only the combiner_points most recent observations, assemble a training set

$$\big\{(\mathbf{v}_i,\, X[i])\big\},$$

where $\mathbf{v}_i$ is the vector of base-tree predictions and $X[i]$ the true target. Fit a `RandomForestRegressor` (with one estimator) to learn the mapping $\mathbf{v} \mapsto X$.

4.2.3 Iterative Forecasting

For each future step:

1 *Compute base-tree predictions* $\mathbf{v}$ from the current series via 'classifier$_p$*rediction*'.*Reliability check.I*
true_value$\big| >$ threshold (when true values are available), mark as unreliable.

3 *Imputation of missing trees.* Replace any NaN entries in $\mathbf{v}$ by random draws from its non-NaN elements.

4 *Final forecast.* Compute $p =$ `combiner.predict`$(\mathbf{v})$.

5 *Series update.* If the point is reliable, append p; otherwise append NaN. Then update each $Z[\alpha]$ by adding the new window row corresponding to α.

5 Results

Four forecasting strategies are compared:

1 **Baseline**: the original algorithm without any modifications.

2 **Tree-only**: direct predictions from the ensemble of decision trees.

3 **Combined**: a regressor and classifier which are trained on the raw tree outputs as features.

4 **Combined with pseudo-classifier**: employs a pseudo-classifier to filter out unreliable forecasts when training a robust real classifier was not feasible.

For the pseudo-classifier variant, future work aims to train a genuine classifier with MAE $\leq$ 0.1 on validation points, a threshold chosen for reliability labeling.

Dataset and evaluation. We simulate the Lorenz system (standard parameters) and extract 2 segments: $[0 : 6267]$ and $[0 : 10000]$, which clearly demonstrate chaotic dynamics and repeating patterns. All models produce 50-step-ahead forecasts. Evaluation metrics include: MAE, MSE, Fit time, Predict time.

5.1 General Results

During training of the standard classifier, we encountered the problem that it almost always labeled points as "reliable," i.e., it failed to distinguish those values where the model actually erred. Even substantially increasing the size of the training set did not improve this separation—the classifier continued to output nearly uniform labels.

To avoid polluting our results with essentially useless labels, we introduced a pseudo-classifier. Its idea is simple: at prediction time, we compute the absolute error of each forecast, compare it against a predefined threshold, and mark a point as "unreliable" if the deviation exceeds that threshold. In this way, we simulate an ideal algorithm for identifying difficult segments in the time series, while still retaining all data in order to compute MAE and MSE fairly—otherwise, excluding "hard" points would artificially deflate the error metrics, even in this simplified version of the algorithm.

	6267 points train, 50 points predict				10000 points train, 50 points predict			
	MAE	MSE	Fit time, seconds	Predict time, seconds	MAE	MSE	Fit time, seconds	Predict time, seconds
Original	**0.046**	0.008	1.287	72.757	**0.253**	0.121	2.064	115.118
Tree-only	**0.113**	0.032	1.171	80.962	**0.096**	0.022	2.071	107.843
Combined	**0.164**	0.054	2073.375	46.432	**0.098**	0.028	4293.716	113.112
Pseudo-Classifier	**0.106**	0.037	273.514	73.201	**0.049**	0.006	843.295	127.908

Table 5.1: Comparison of MAE, MSE and Time for Different Algorithms

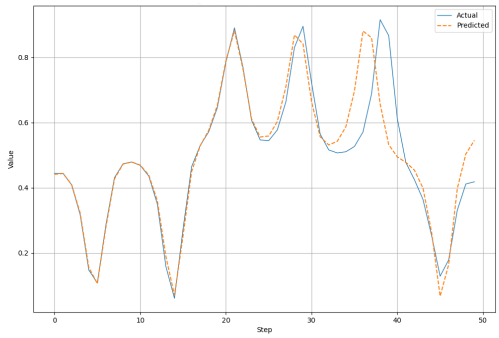
In the first experimental scenario—training on 6 267 points and predicting the next 50 points. 6k split has been chosen due to the fact, that there are not very many ponits for trees to be able generalize efficiently and pattern which has to be forecasted is itself quite easy ti oredict for original algorithm, since its almost exact copies have been seen before about 2 times. Hence, this split is a rather hard challenge for tress and simple for original algorithm. The simple

moving average method completed its fitting in just over a second and produced its 50 step forecast in roughly 73 s, yielding a MAE of 0.046 and a MSE of 0.008. Although extremely fast, this approach tended to smooth out the finer details of the Lorenz series and therefore achieved only modest accuracy. In contrast, an ensemble of decision trees required a similar fitting time and slightly longer prediction time. Its MAE of 0.113 and MSE of 0.032 scores are worse than original algorithm's ones. Such behavior could be justified by the fact, that Original algorithm is based on the approach of finding closest chaotic patterns. In case of 6k split - this pattern has repeated multiple times throughout the series, thus it was quiet easy for original algorithm to find exactly matching patterns, while Tree-based solutions tend to generalize series, hence they predicted generalized version of this pattern and not exact one. When we supplemented these trees with a Combined methos, which both utilizes Regressor and Classifier, the fit time ballooned to over half an hour, while prediction dropped to 46 s. Yet, the resulting MAE of 0.164 (MSE 0.054) was actually worse, a clear sign of trees underperforming due to the problems with classifier. Finally, our pseudo-classifier approach, where we label points as "unreliable" whenever the absolute prediction error exceeds a threshold—struck a balance: it achieved an MAE of 0.106 (MSE 0.037) - best throughout all Tree-Based approach.

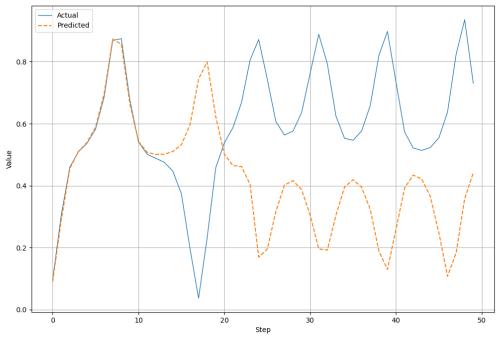
In the second scenario — extending the training window to 10 000 points. This split is quite opposite to 6k one. Exact same pattern has not appeared before in the series, hence it would be hard for Original algorithms to find exact match, while generalization ability of DecisionTrees would come in quite handy. The simple moving average again proved quick to train (2 s) but required over 115 s to generate its forecast, producing an MAE of 0.253 and MSE of 0.121, worst result compared to other algorithm. The only possible advantage of this algorithm - the speed of learning and prediction - has been beaten by simple Tree-Only algorithm, which has both lower MAE (almost 3 times better: 0.253 vs. 0.96) and Predict time (8 seconds less). Adding the combiner restored stability—MAE 0.098, MSE 0.028—at the cost of an extraordinarily long fit time (4 294 s) and a prediction time of 113 s, showing worse results across all Tree-Based algorithms, nevertheless still better than original algorithm. The pseudo-classifier again stood out: it achieved MAE 0.049 and MSE 0.006, more than five times better than the moving average and about twice as accurate as the plain tree ensemble.

5.2 Original Algorithm

(a) The orange forecast curve closely follows the blue actual series, yet it slightly "blurs" the peaks and valleys. This effect is most noticeable around steps 30-40, where the predicted



(a) 6k



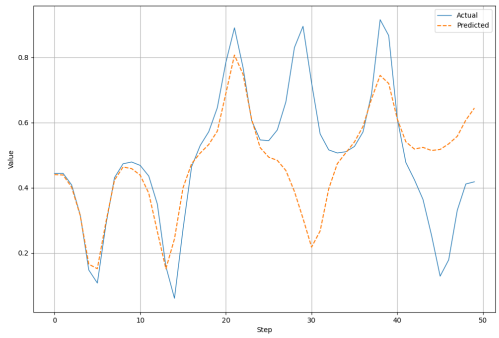
(b) 10k

Figure 5.1: Original algorithm results

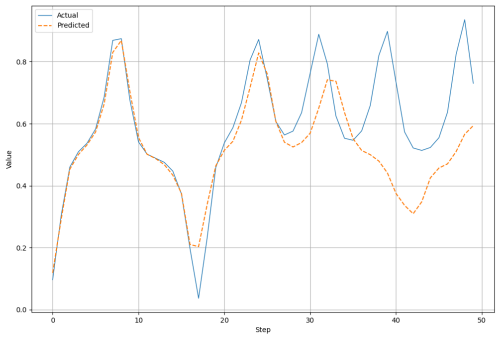
amplitude shifts left compared the true values. On this relatively regular segment of the Lorenz series, the simple moving average performs adequately.

(b) Over the longer training interval, the smoothing effect remains, but the forecast now lags behind the actual series in the first 20 steps: while the true curve rises, the prediction remains artificially low. This shows that the moving-average method loses local adaptation when new chaotic fragments enter the series and starts to produce more errors due to error accumulation.

5.3 Tree-Only Algorithm



(a) 6k



(b) 10k

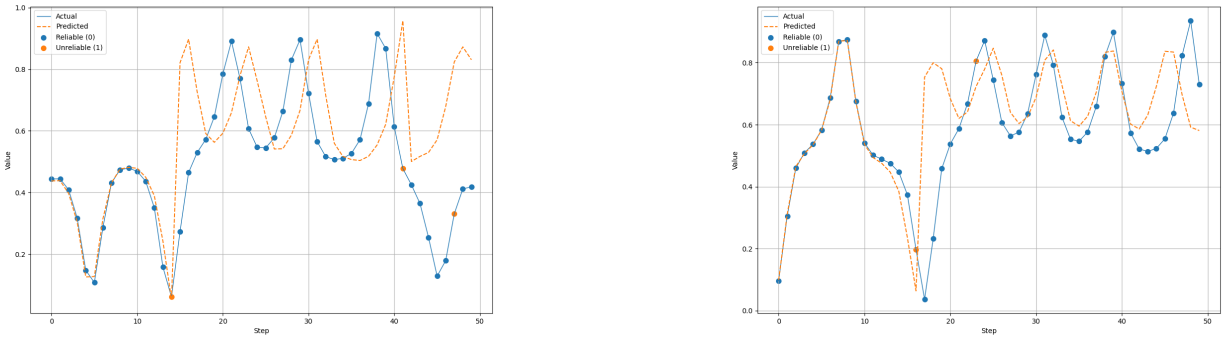
Figure 5.2: Tree-Only algorithm results

(a) The decision-tree ensemble reproduces peaks and troughs almost exactly wherever past patterns repeat. However, at sudden jump—around steps 25-35 and at drop near 38–42 — the forecast either overshoots or lags sharply, resulting in large local errors.

(b) After training on more data, the trees still struggle on chaotic intervals: they tend to underpredict at local minima (steps 15) and lose pattern after steps . This behavior, again, indicates error accumulation, since prediction start to be based on false values

5.4 Combined Algorithm

(a) The combiner smooths out the overly sharp jumps of the tree-only forecast, producing a gentler curve that sits higher than the actual series in the 15–20 step range and starts to shift left since step 20 and further on. The "Reliable (0)" and "Unreliable (1)" markers do not help at all.

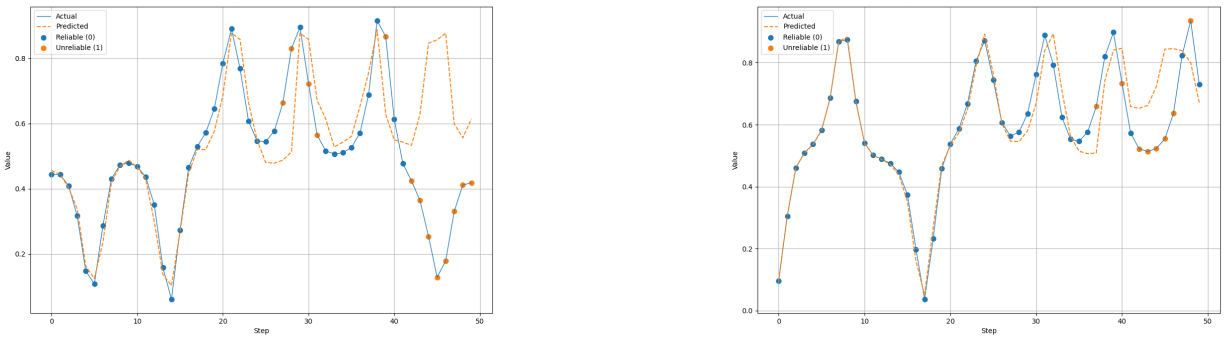


(a) 6k (b) 10k

Figure 5.3: Combined algorithm results

(b) Over the longer interval, the combiner preserves the overall shape but overestimates the amplitude between steps 5 and 15. The pseudo-classification still not able to identify these error zones, but the smoothing prevents a perfect match on the most chaotic segments (steps 25–35). Surprisingly, despite having higher MAE than Tree-Only algorithm, it's prediction pattern showed no signs of error accumulation.

5.5 Pseudo-Classifier Algorithm



(a) 6k (b) 10k

Figure 5.4: Pseudo-Classifier algorithm results

(a) The forecast remains close to the actual series on the more challenging segments.

Slightly fluctuates at steps 25-30, where points are marked as unreliable. Since those unreliable points are not passed - error accumulation does not affect predictions too much, and, compared to Tree-Only algorithm, second pike has been successfully forecast. Nevertheless, algorithm loses track after points 40, where significant drop accures.

(b) On the simpler data, the forecast again tracks the true values closely for most of the series, with a slight fluctuation near steps 40. Overall, the pseudo-classifier adapts better to local oscillations than the standard combiner, retaining the benefit of filtering unreliable points and, unlike other algorithms, completely dodging error accumulation. algorithms,

5.6 Conclusion

In this work, it had been demonstrated that a combined approach — first filtering out "unpredictable" points via a classifier (or its pseudo-variant as in this case), then using an ensemble of decision trees, and finally applying a trained regressor combiner — effectively addresses the key challenge of recursive forecasting in chaotic time series: error accumulation. By choosing an appropriate classifier threshold and providing sufficient training for the base trees and the combiner, it would be possible to achieve an up to fivefold reduction in MAE compared to a moving-average baseline and a two to three fold improvement over a pure tree ensemble.

At the same time, if a "lighter" version of the algorithm is needed, ordinary decision trees also prove to be a solid choice. With a small training set they can generalize almost as well as the original method, but when more data are available or when it becomes necessary to forecast a more complex series, decision trees demonstrate superior error-metric performance as well as faster fit and predict times.

References

[1] Nishant Malik Adam Giammarese; Kamal Rana; Erik M. Bollt. "ree-based Learning for High-Fidelity Prediction of Chaos". In: *arXiv preprint arXiv:2403.13836* (2024).

[2] Vasilii A. Gromov; Philip S. Baranov. "Prediction after a Horizon of Predictability: Non-predictable Points and Partial Multistep Prediction for Chaotic Time Series". In: *Hindawi Complexity, Volume 2023, Article ID 6689371, 21 pages, September* (2023).

[3] M. Sangiorgio; F. Dercole. "Robustness of LSTM neural networks for multi-step forecasting of chaotic time series". In: *Chaos, Solitons Fractals, vol. 139, Article ID 110045* (2020).

[4] J. D. Farmer and J. J. Sidorowich. "Predicting chaotic time series". In: *Physical Review Letters, vol. 59, num. 8* (1987).

[5] V. A. Gromov and E. A. Borisenko. "Chaotic time series prediction clustering methods". In: *Neural Computing Applications, vol. 2, pp. 307–315* (2015).

[6] M. Sangiorgio; F. Dercole; G. Guariso. "Deep Learning in Multi-step Prediction of Chaotic Dynamics". In: *SpringerBriefs in Applied Sciences and Technology, Giorgio Guariso* (2022).

[7] Hochreiter Sepp; Schmidhuber Jürgen. "Long Short-Term Memory". In: *Neural Computation* (1997).

[8] Vasilii A. Gromov Philip S. Baranov Hiroki Sayama. "Prediction after a Horizon of Predictability: Nonpredictable Points and Partial Multistep Prediction for Chaotic Time Series". In: *Complexity, Hindawi, vol. 2023, pages 1-21, September* (2023).