Evaluating Simplification Algorithms for Interpretability of Time Series Classification

Felix Marti-Perez a,*, Brigt Håvardstun b,**, Cèsar Ferria, Carlos Monserrata and Jan Arne Telleb

^aVRAIN, Universitat Politècnica de València, València, Spain ^bDepartment of Informatics, University of Bergen, Bergen, Norway

Abstract. In this work, we introduce metrics to evaluate the use of simplified time series in the context of interpretability of a TSC - a Time Series Classifier. Such simplifications are important because time series data, in contrast to text and image data, are not intuitively understandable to humans. These metrics are related to the *complexity* of the simplifications - how many segments they contain - and to their *loyalty* - how likely they are to maintain the classification of the original time series. We employ these metrics to evaluate four distinct simplification algorithms, across several TSC algorithms and across datasets of varying characteristics, from seasonal or stationary to short or long. Our findings suggest that using simplifications for interpretability of TSC is much better than using the original time series, particularly when the time series are seasonal, non-stationary and/or with low entropy.

1 Introduction

Temporal data is encountered in many real-world applications ranging from patient data in healthcare [23] to the field of cyber security [30]. Deep learning methods have been successful [15, 23, 30] for TSC - Time Series Classification - but such methods are not easily interpretable, and often viewed as black boxes, which limits their applications when user trust in the decision process is crucial. To enable the analysis of these black-box models we revert to post-hoc interpretability. Recent research has focused on adapting existing methods to time series, both specific methods like SHAP-LIME [9], Saliency Methods [14] and Counterfactuals [4], and also combinations of these [26].

As humans learn and reason by forming mental representations of concepts based on examples, and any machine learning model has been trained on data, then we believe that data e.g. in the form of prototypes and counterfactuals is indeed the natural common language between the user and this model. However, the basic problem is that compared to images and text, time series data are not intuitively understandable to humans [29]. This makes interpretability of time series extra demanding, both when it comes to understanding how users will react to the provided explanations and to predict what explanatory tools are best. For example, in [12] a tool was given for explainability of a TSC, that allowed model inspection so the user could form their own mental model of the classifier. However, a user evaluation showed that non-expert end-users were not able to make

Theissler et al [33] give an intriguing taxonomy of XAI for TSC, divided into those focused on (i) Time-points (e.g. SHAP and LIME) or (ii) Subsequences (e.g. Shapelets) or (iii) Instances (Prototypes, CF, Feature-based). Explaining a classifier by instances like prototypes has a definite appeal, but it is a challenge how to highlight features important for the classification, while at the same time simplifying the instance to mitigate the non-intuitive aspects of this domain. Theissler et al [33] review the literature on XAI for TSC and of the 9 papers they discuss on Instance-based explanations using Prototypes or Features, it is remarkable that not a single one is specialized for Local explanations, rather they are all Global. The algorithms we evaluate in this paper fill this gap, as they are ways of *simplifying* a time series by straight-line segments¹

We introduce metrics to evaluate the use of algorithms computing simplified time series in the context of interpretability of a TSC, a Time Series Classifier. Such simplification algorithms are important because time series data, in contrast to text and image data, are not intuitively understandable to humans. These metrics are related to the *complexity* of the simplifications produced by the algorithms - how many segments they contain as a percentage of the original - and to their *loyalty* - how likely they are to maintain the classification of the original time series. We employ these metrics to evaluate four distinct simplification algorithms, across several TSC algorithms and across datasets of varying characteristics, from seasonal or stationary to short or long.

In the rest of this paper we first discuss related work and cover some standard definitions before presenting the metrics we will use, the four simplification algorithms we will evaluate, the TSCs we will employ and the 28 UCR datasets with their varying characteristics. We then show the results of our experiments, first comparing the four algorithms over a variety of dataset characteristics, and then focusing on the best performing algorithms and evaluating how useful they are for interpretability. We include also an evaluation using multimodal large language models. Our findings suggest that using simplifications for interpretability of TSC is much better than using the original time series, particularly when the time series are seasonal, non-stationary and/or with low entropy.

use of this freedom, supporting the notion that time-series are particularly non-intuitive for humans.

^{*} Corresponding Author. Email: fmarper@upv.edu.es, Supported by the Norwegian Research Council, project Machine Teaching for XAI.

^{**} Corresponding Author. Email: Brigt.Havardstun@student.uib.no

Piecing together several such local explanations, say for prototypes of each class, this can still form a global explanation of the given TSC.

2 Related Work

Several techniques have been applied to generate explanations from TSC models [33, 25]. Specifically, techniques previously used on Convolutional Neural Networks or Recurrent Neural Networks have been applied for TSC. For instance, the authors in [26] apply to time series several XAI methods previously used on image and text domain. They also introduce verification techniques specific to times series, in particular a perturbation analysis and a sequence evaluation.

Another alternative is to produce explanations through examples, and these can be specifically utilised in the time series domain. One type of this explanation method involves giving the nearest example from the training dataset that acts as a prototype to depict the normal behaviour of a similar sample [8]. A method to generate prototypes for time series data using an autoencoder is presented in [7]. The main novelty of the work is the method to generate diverse prototypes.

Given that in many cases raw time series can be too complex for humans, several studies have tried to employ simplified versions as explanations. In [16], Keogh et al propose a dimensionality reduction technique for time series, called Piecewise Aggregate Approximation, as a tool for similarity search in large time series databases. In [18], a comprehensive survey on time series segmentation, the authors highlight that the optimal number of segments often depends on the underlying patterns in the data and the specific objectives of the analysis. Also, in [1] Camponogara and Nazari introduce a range of piecewise-linear models and algorithms for unknown functions. Another option is to segment time series into fixed-width windows and employ these to justify decisions. In [27] Schlegel et al propose TS-MULE, a model explanation method working to segment and perturb time series data and extending LIME. A study on the effect of segmentation on time series, in particular in the field of finance for the use of pattern matching was presented in [35]. In [28], Si and Yin also apply segmentation to financial time series data, now as a preprocessing step to locate technical patterns. Similarly, in [37], the authors employ financial time series segmentation and introduce the Optimal Multi-Segment Linear Regression (OMSLR) algorithm. Their approach aims to minimise the global mean square error (between the simplification and the raw TS) for a given number of segments k.

3 Simplifications and Metrics

We first recall basic notions and then introduce the simplifications and metrics that we will study in this paper.

Staying consistent with earlier notation [33, 4] a time series $T=\{t_1,t_2,\ldots,t_n\}$ is an ordered set of n real-valued observations (or time steps). Note we may also view each $t_i=(x_i,y_i)$ as a pair consisting of an x-value (the time) and a y-value (the observation), where $x_i< x_{i+1}$, and often $x_i=i$. A time series dataset $D=\{(T_1,c_1),(T_2,c_2),\ldots,(T_m,c_m)\}\in R^{m\times n}$ is a collection of such time series where each time series T_i has a class label c_i . Thus, for binary classification tasks we would have $c_i\in\{0,1\}$. Given such a dataset D, Time Series Classification (TSC) is the task of training a mapping C_D from the space of possible inputs to a predicted class. Interpretability of TSC concerns giving a human user an understanding of how such a trained TSC decides on a predicted class. Exemplar-based interpretability employs showing examples of classifications made by the TSC, and this will allow the user to form their own mental model of the classifier. Prototypes are time series

exemplifying the main aspects responsible for a classifier's specific decision outcome. Showing prototypes is well-known in exemplar-based interpretability.

For TSC interpretability, in particular for long time series, such prototypes often have too much information, and we therefore focus on what we call simplifications. For the purposes of this paper a simplification of a time series $ts = \{t_1, t_2, \ldots, t_n\}$ on |ts| = n time steps consists of dropping some of the time steps, thus keeping only time steps at a subset $S \subseteq \{1, 2, \ldots, n\}$ of time points and replacing the rest by the points lying on the straight-line segments given by the kept points, thus resulting in a new time series $sts = \{t'_1, t'_2, \ldots, t'_n\}$ with $t'_i = t_i = (x_i, y_i)$ for $i \in S$ and otherwise $t'_i = (x_i, y'_i)$ with y'_i the value at x_i of the straight line between t_j and t_k for j < k and $\{j, j+1, \ldots, k\} \cap S = \{j, k\}$. We will say that the simplification sts consists of |S| - 1 segments, and we denote its number of segments by ||sts||. See in Figure 1 an example showing a simplification with 4 segments obtained by keeping 5 time steps (at times 0,9,13,19,23) of an original time series on 24 time steps (at times 0 to 23).



Figure 1: An original time series (dotted line) and the simplification on 4 segments produced by RDP with $\alpha=0.2$.

A simplification algorithm for time series called A takes as input an original time series ts and outputs a simplification A(ts) on ||A(ts)|| segments. Note that the length of A(ts) is the same as the length of ts so a classifier trained for time series of that length can be used on the simplification A(ts).

Definition 1. To evaluate the usefulness of a simplification algorithm A for interpretability of a TSC C_D , we introduce the following metrics. Let P be a representative subset of inputs, chosen randomly according to the expected distribution of inputs to C_D , and let these original inputs have length n:

- Complexity: $: \frac{1}{n} \frac{1}{|P|} \sum_{ts \in P} ||A(ts)|| + 1$, the expected number of time points kept in the simplification, as a fraction of the length of the original
- Loyalty: $\frac{1}{|P|}|\{ts \in P : C_D(ts) = C_D(A(ts))\}|$, the expected probability that the simplification has the same classification as the original
- Kappa loyalty: $\frac{p_0-p_e}{1-p_e}$ where p_0 is the relative observed agreement, and p_e is the hypothetical probability of chance agreement

Thus complexity is a value between $\frac{2}{n}$ (when all simplifications of instances in P are on a single segment) and 1 (when all simplifications are the trivial one which does not remove any time step from the original), while loyalty is a value between 0 (when all simplifications are classified by C_D as different from the original ones) and 1 (when complete agreement, all simplifications are classified the

same as the original ones). The Kappa loyalty [2] will have a maximum value of 1 when we have complete agreement, a value of 0 if the agreement is no better than chance, and below 0 if a tendency for different classes. Cohen's Kappa was used as a performance metric as it accounts for agreement occurring by chance, providing a more robust measure than accuracy, particularly in the presence of class imbalance. Note that when using a simplification algorithm for interpretability of TSC, showing the simplifications to a user, then low complexity allows a more easy focus on important features, and high loyalty minimizes the mistakes accrued from not showing the originals.

4 Simplification Algorithms

We briefly describe the four simplification algorithms we have chosen to focus on. Although they are quite distinct, they share a number of properties, which will allow for a fair comparison:

- \bullet their input consists of an original time series, given by n points
- their output is a simplification given by a subset of input points
- they have a parameter α and increasing its value will monotonically decrease number of segments in output, from n-1 to 1.

4.1 Ramer-Douglas-Peuker

RDP is a recursive algorithm introduced in [24, 6]. Given n points $p_1,...,p_n$ select a point p_d with the largest distance to the straight line between the first point p_1 and last point p_n . If the distance to p_d is no more than α , then we are in a base case and return the two points p_1 and p_n . If the distance to p_d exceeds α , then recursively process the two sequences of points formed by $p_1,...,p_d$ and $p_d,...,p_n$. The implementation ([10]) used for RDP has runtime $O(n \log n)$.

4.2 Visvalingam—Whyatt algorithm

VW is an iterative algorithm introduced in [34]. It computes areas of triangles formed by successive triples of points a,b,c; in each iteration, the middle point b with the smallest associated triangle is removed, the area of neighbouring triangles is recomputed, and the process is repeated. The process stops when the area of the smallest triangle exceeds α . The implementation of VW we use ([13]) has runtime $O(n \log n)$.

4.3 Bottom-Up

BU is based on an iterative algorithm introduced in [17] with the same name. We slightly modified the algorithm from [17] so that it adheres to the simplification algorithms we study here - selecting subsets of original time steps. Initially each time point is viewed as a trivial seg (a special segment), and in each iteration we merge two consecutive segs A and B (if A ends in time step k then B starts in time step k+1) with the merged seg AB going from the start of A to the end of B (in original BU the segment AB would be the Least Square Fit over this area). The two segs merged into some AB in any iteration should have lowest error, which is the sum over each time point t within AB, of the absolute value of the difference between the value of the original time series at t and the value of the AB line at t. We stop when the lowest error exceeds α , with s segs $s_1, s_2, ..., s_s$ such that if s_i ends in s_i then s_i that is s_i and we return the corresponding simplified time series of length s_i . While the resulting

simplification has only s segs, if we add the gaps between consecutive segs we get 2s-1 ordinary segments. Our implementation of BU has runtime $O(n \log n)$.

4.4 Optimal Simplification

OS is a dynamic programming algorithm introduced in [32]. It finds the piecewise linear simplification (extending the first and last of its segments to the start and end respectively) that minimizes the sum of: Squared Euclidean Distance to the original time series plus α times the number of segments used. Our implementation of OS has runtime $O(n^3)$. Note that, in contrast to the other three algorithms, it finds the optimal simplification for a given α , rather than just being a greedy algorithm, but it is also more time consuming.

4.5 Normalized complexity parameter

To ensure uniformity we alter each of the four simplification algorithms so that it has a normalized complexity parameter called α_c . This parameter will range between a minimum of 0 (corresponding to a high value of α giving a small number of segments as output) and a maximum of 1 (corresponding to a low value of α giving a high number of segments as output).

5 Datasets and Classifiers

5.1 Datasets Selection

The UCR repository [3] contains 128 datasets for univariate TSC. Their features cover problems from binary up to 52-class classification, with time series lengths ranging from 15 to 2,700 data points. Given that the datasets come from a wide range of domains, such as sensors, medical devices, or traffic, the characteristics of these datasets is diverse.

The datasets come with predefined splits for training, validation, and test sets. In all our experiments, we utilise those predefined splits.

To keep runtime manageable, we included only those UCR series of length less than 200 data points. This has resulted in 28 datasets of varying length, class and domain. See Table 4 for details on the 28 chosen datasets.

Additionally, we categorise each dataset by stationarity, seasonality, and entropy, enabling in-depth analysis of how different simplification algorithms perform under varying data characteristics.

5.2 Feature Characterisation

The Augmented Dickey Fuller (ADF) test [5] is a statistical test used to determine the presence of a unit root in the series, thus determining if a time series is non-stationary.

We set the significance level at 0.05 and apply the test to each instance: if the p-value is below it, we reject the null hypothesis that a unit root is present in the time series and label that instance "stationary"; otherwise, we label it "non-stationary." At the dataset level, we declare a dataset "stationary" if at least 80% of its instances are stationary (supermajority), "non-stationary" if at most 50% are stationary (simple-majority), and "partially-stationary" when the proportion lies between these thresholds [36].

The autocorrelation function (ACF) measures the correlation of a time series with a lagged version of itself. We utilise an extension that uses Fast Fourier transform (FFT) convolution to make ACF faster and more suitable for longer time series. A dataset is considered to be seasonal if most of its instances have a correlation coefficient greater

than 0.4, thus being seasonal. This cut-off was decided empirically by visually inspecting the time series.

Approximate entropy [21] is a technique used to quantify the unpredictability of fluctuations in time series. This returns a continuous value between 0 (low entropy) and 1 (high entropy), which will determine the entropy of the time series. To obtain the entropy of a dataset, we obtain the mean entropy over all instances.

To facilitate the analysis, we discretise the entropy scores into three equal-frequency bins. We determine the breakpoints to be 0.23 and 0.27. Instances with entropy up to 0.23 are labelled as low entropy, those between 0.23 and 0.27 as medium entropy, and those above 0.27 as high entropy.

5.3 Classifiers

Different types of classification algorithms are used for TSC. Throughout the literature, the most common algorithms are:

- Logistic Regression: A linear model that estimates the probability of each class using weighted input features; it is simple, interpretable (white-box), and best suited for linearly separable time series. We use scikit-learn's LogisticRegression [20] with its default L2 penalty and solver.
- Decision Trees: A non-linear, rule-based model (white-box) that recursively partitions the feature space to make decisions. Capable of handling complex relationships and capturing non-linearities in time series, but prone to overfitting. Implemented via scikit-learn's DecisionTreeClassifier [20] using the Gini impurity criterion and no maximum-depth constraint.
- k-Nearest Neighbour (kNN): A non-parametric method that classifies a time series based on the majority label among its closest examples in the training set. It can be interpretable (white-box), but its decisions can become less transparent with higher dimensionality. We leverage tslearn's KNeighborsTimeSeriesClassifier [31], configured with k = 5 and distance-weighted voting.
- Convolutional Neural Networks (CNN): A deep learning architecture that automatically extracts and learns hierarchical features through convolutional filters is highly effective for discovering patterns in time series data. However, it is considered a black-box due to its complex and opaque decision processes. Our networks are built in PyTorch [19] consisting of three 1D convolutional layers, a global pooling layer, and a final fully-connected classification layer; trained with the Adam optimizer and cross-entropy loss.

We first check all four classification algorithms to see which is most suited. Prior to training, we normalize every split of the dataset (training, validation and test sets). We then fit each model on the normalized training data using default hyperparameters. We evaluate these four widely-used TSC algorithms, which span the spectrum from simple, interpretable white-box models to black-box deep learners. The accuracy on the validation set shows that CNN emerges as the best-performing model across our benchmark.

CNNs are thus both the top performers and the main black-box models where interpretability is key, and for simplicity we focus solely on them in the following sections, as they will be the most relevant and interesting for applying simplifications.

6 Experiments and Results

It follows from the discussion in Section 3, where we introduced our metrics, that for interpretability of a classifier C_D we are seeking

simplification algorithms that have low complexity and yet at the same time high loyalty, or rather high kappa loyalty that takes into account any imbalance in the distribution between classes. In the first part of this section we investigate which of the four simplification algorithms introduced in Section 4 have the best complexity versus loyalty performance, and we will do this over a variety of characteristics of datasets D as described in Section 5.2, while focusing on the CNN classifiers as explained in Section 5.3. In the last part of this section we focus on the simplification algorithms having the best performance and consider if the actual values we get for loyalty versus complexity are good enough to be used in an interpretability situation

6.1 Comparing four algorithms

For the experiments we have 4 simplification algorithms with normalized complexity parameter α_c , and we also have a separate CNN classifier C_D for each of the 28 normalized time series datasets D. For each pair of a simplification algorithm A and a classifier C_D (for a dataset D) we want to evaluate the interplay between loyalty and complexity, and do this as follows.

- Pick 100 instances P at random from the dataset D, preserving the original fraction of instances from each class
- Loop over 100 values of the complexity parameter α_c, from 0 to 1 in steps of 0.01.
- For each instance $ts \in P$ and each value of α_c , we run A to get the simplification $sts = A(\alpha_c, ts)$.
- We record two things:
- (||sts|| + 1)/|ts|, to compute complexity
- $C_D(ts) == C_D(sts)$, (True or False) to compute loyalty
- For each value of α_c, over all 100 ts ∈ P, we then compute the complexity, loyalty and kappa loyalty as per Definition 1:
 - the average complexity in the 100 simplifications having this value of α_c
 - the percentage of loyal simplifications, also called agreement
 - Cohens kappa value for the loyalty agreement (which takes into account unbalanced classes)

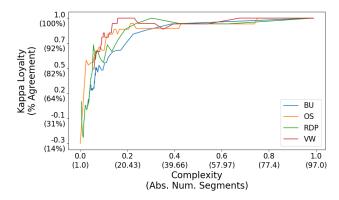


Figure 2: ECG200 dataset: Plot of Kappa loyalty (percent loyalty in parenthesis) versus Complexity (average number of segments in parenthesis) of the CNN classifier, for all 4 simplification algorithms.

After doing the above we have, for each triple (A, α_c, C_D) of algorithm, complexity parameter and classifier, 100 values of kappa

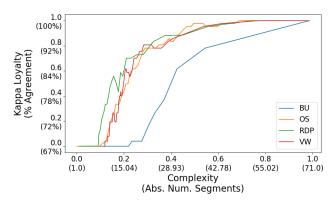


Figure 3: SonyAIBORobotSurface1 dataset: Plot of Kappa loyalty (percent loyalty in parenthesis) versus Complexity (average number of segments in parenthesis) of the CNN classifier, for all 4 simplification algorithms.

loyalty and complexity, one for each instance in P. For each algorithm and classifier pair (A,C_D) , to visualize the results of these 10.000 simplifications (100 instances in P times 100 distinct values of α_c) we plot a curve. See Figures 2 and 3 showing kappa loyalty (loyalty in parenthesis) versus complexity (number of segments in parenthesis) for all 4 simplifications algorithms on two different CNN classifiers C_D on two datasets.

To compare the performance of the four simplification algorithms, we measure the AUC - Area Under Curve - as a value between 0 and 100, of the 28 such kappa loyalty versus complexity curves for each simplification algorithm. In Table 1 we show in the first row the average AUC for each of the four algorithms, and it is clear that OS and R have the best performance on average. The table also shows the average performance on classifiers for datasets of various characteristics, like Number of Classes, Stationarity, Seasonality and Entropy.

Table 1: AUC values for the 4 algorithms on the CNN classifiers. In the top row (Mean) we see that on average over all 28 datasets OS and RDP achieve the highest values. The other rows show an average over datasets of different characteristics. Datasets that are Non-Stationary, Seasonal and with Low Entropy achieve significantly higher values than those that are not.

Metric	OS	RDP	BU	VW	Datasets
Mean	88.3	87.3	79.9	84.7	28(all)
Binary	87.9	87.1	79.9	84.9	11
Multiclass (>2)	88.6	87.4	79.9	84.6	17
Stationary	83.7	80.4	69.1	79.3	4
Non-Stationary	89.3	88.5	82.0	85.5	22
Partially-Stationary	85.1	86.2	77.1	84.3	2
Seasonal	94.3	92.9	88.0	91.5	6
Non-Seasonal	86.4	85.5	77.4	82.5	22
Low-Entropy	91.7	90.8	85.1	87.7	10
Medium-Entropy	86.7	84.8	77.6	82.1	9
High-Entropy	86.1	85.6	76.5	83.7	9

We see that datasets that are Non-Stationary, Seasonal and with Low Entropy achieve significantly higher values, and in Table 2 we focus on the four datasets sharing all these characteristics. Remarkably, at loyalty value 0.95 the best performing algorithm never has complexity higher than 0.05, meaning that on datasets with these characteristics we are able to keep at most 5% of the original time points and still have a simplification that has the correct classification with at least 95% probability.

Table 2: Average AUC values for the CNN classifiers on the four Datasets that are Non-Stationary, Seasonal, and with Low Entropy. In parenthesis we show the average complexity at loyalty 0.95. All algorithms have high AUC, and low complexity, with RDP and OS the best performers.

	OS	RDP	BU	VW
GunPoint AgeSpan	97.6 (0.04)	97.6 (0.03)	95.8 (0.07)	95.6 (0.04)
GunPointOld VersusYoung	97.7 (0.04)	98.6 (0.02)	97.4 (0.03)	95.6 (0.04)
BME	95.9 (0.06)	97.3 (0.03)	93.2 (0.03)	93.2 (0.06)
UMD	94.9 (0.05)	96.0 (0.06)	89.5 (0.12)	93.3 (0.11)

We also measure performance of the four algorithms at fixed loyalty values of 0.8, 0.85, 0.9 and 0.95. In Table 3 we see that again OS and RDP have the best performance, with a resulting best average complexity of between 0.12 and 0.22, which means that the time points kept in the simplification at these high loyalty values is on average between 12% and 22%. This is a strong indicator that simplifications are useful in interpretability of TSC.

Table 3: Average Complexity values over all 28 datasets for the 4 algorithms on the CNN classifiers when setting loyalty (percent agreement/100) to 0.8, 0.85, 0.9 and 0.95.

Loyalty	OS	RDP	BU	VW
0.8	0.12	0.12	0.19	0.14
0.85	0.14	0.15	0.24	0.15
0.90	0.17	0.19	0.30	0.19
0.95	0.22	0.26	0.39	0.25

In Table 4 we show the average number of segments of the simplifications, at these 4 loyalty values, for both OS and RDP, for all 28 datasets.

6.2 A closer look at OS and RDP

We have seen that the OS algorithm has the best average performance. However, being a cubic algorithm it has a time complexity that makes it unsuitable for very long time series, and so we focus also on RDP which has the best average performance out of the three faster $O(n \log n)$ algorithms. For these two algorithms, OS and RDP, we consider the following question over a variety of dataset characteristics: for exactly what values of loyalty is the complexity low enough that the user will benefit from seeing the simplifications?

We have been experimenting with a tool for interpretability of a TSC \mathcal{C}_D using the OS and RDP simplification algorithms. See a screenshot in Figure 4.

The user selects an algorithm, say RDP, and a loyalty value, say 90%, and the tool will first find the minimum α_c value so that RDP with this parameter value has expected loyalty at least 90%. In the next step, prototypes for the classification of dataset D by C_D are computed, say two sets P_0 and P_1 for D being binary. In the final step, instead of showing these prototypes to the user, we show the simplifications $RDP(\alpha_c,ts)$, for each ts in T_0 and T_1 , together with its class. This way the user can make his or her own mental model of how C_D makes its classification. As stated above, the main question is if the number of segments in these simplifications is small enough that the user is willing to pay the price of the loyalty being only 90%. Clearly, this will depend on the length of the original time series, with shorter time series having fewer segments in their simplifications. In Figures 5 and 6 we address this question, by showing complexity (fraction of time points kept in the simplification) across



Figure 4: Screenshot of a tool where the user can select a loyalty threshold and get a simplification, using a chosen algorithm, having the fewest number of segments and expected loyalty meeting the threshold.

various times series lengths and how this varies as the loyalty values vary, for OS and RDP respectively. For a significant number of datasets the resulting number of segments is below 10, at least for loyalty below 95%, also for some of high length. This is another indicator that simplifications are useful in interpretability of TSC.

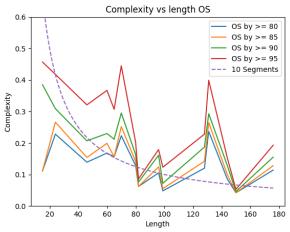


Figure 5: OS Algorithm: Complexity versus time series length for 4 loyalty values from 80% to 95%, over all 28 datasets. The dotted line shows where simplifications on 10 segments would lie.

7 Interpretability of TCS by MLLMs

To demonstrate the practical application of our work, we employed the simplification algorithms to investigate whether simplified (yet loyal) versions of prototype time series enable large language models to more easily deduce the classification of a binary TSC. Recent studies have demonstrated the potential of multimodal large language models (MLLMs) to analyse time series data and perform tasks such as classification [22] and anomaly detection [38]. Although the simplifications introduced in this paper are meant primarily as explanations for human users we evaluate them as explanations for MLLMs.

In our experiment, we focus on binary classification tasks by teaching GPT-4.1 using simplified prototype examples from both the positive and negative classes as training input, and subsequently prompting it to classify unseen simplified representations of test instances. KMediod clustering using Dynamic Time Warping (DTW)

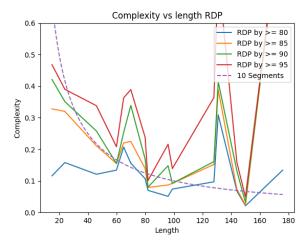


Figure 6: RDP Algorithm: Complexity versus time series length for 4 loyalty values from 80% to 95%, over all 28 datasets. The dotted line shows where simplifications on 10 segments would lie.

as the distance metric is utilised to obtain resulting medioids as the prototypes of each class [11]. We measure the accuracy of the MLLM by the fraction of test instances where the MLLM classifies the simplified time series the same as what the CNN TSC classifies the original time series. We test the learning capacity of the MLLM for different levels of simplification, employing the OS algorithm.

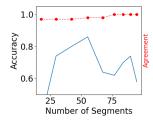
Our hypothesis is the same as we have been arguing in earlier sections of this paper. As we initially increase the number of segments in the simplifications used to teach and test we should see an increase in the MLLM accuracy because the loyalty of the simplifications is increasing. However, when the number of segments becomes too high the MLLM accuracy should drop because it becomes harder to identify the important features, even though the loyalty is even higher.

Specifically, we provide 4 positive prototype examples and 4 negative prototype examples and 50 test examples in batches of 10. This is done for each of the 8 binary datasets in Table 4. On average, the maximum accuracy was 0.82 achieved at average complexity of 0.39, thus keeping an average of less than 40% of the original time points.² This process has been applied to all the TSCs of two classes present in the dataset used (see section 5). Figure 7 shows two examples of the behaviour observed in almost all MLLM experiments. In Figure 7, we plot the accuracy of the MLLM depending on the number of segments of the simplifications, for the ECG200 and SonyAIBORobotSurface1 datasets. In both cases, we see that the number of segments has a direct effect on the learning capacity of the MLLM. The highest accuracy is in both cases 0.86 and is obtained with 26 segments for SonyAIBORobotSurface1 and with 53 segments for ECG200. We see that full complexity (raw time series) provides too much information to the learner, while a drastically simplified time series did not obtain positive results since crucial information is missing. In Figure 7 we also see the loyalty, which in this range is always high for ECG200, and note that as expected the accuracy is always below the loyalty and that for SonyAIRobotSurface1 the highest accuracy is in the vicinity of the knee of the loyalty curve.

² The employed prompts is provided in Appendix A.

Table 4: The 28 UCR datasets. The first 5 columns denote Cl. (class length), Len. (length of time series), Stat. (stationarity), Seas. (seasonality) and Entr. (entropy). The next 4 show average number of segments in the simplifications produced by OS, on the 100 instances chosen randomly, for loyalty of 80 to 95. The last 4 columns show the same for RDP.

Name	Cl.	Len.	Stat.	Seas.	Entr.	OS 80	OS 85	OS 90	OS 95	RDP 80	RDP 85	RDP 90	RDP 95
Adiac	37	176	Yes	Yes	0.27	20	22	27	34	24	62	100	138
BME	3	128	No	Yes	0.12	6	6	7	8	2	3	3	4
CBF	3	128	No	No	0.83	23	28	32	35	15	16	18	23
Chinatown	2	24	No	No	0.23	5	6	6	7	8	8	9	9
DistalPhalanx OutlineAgeGroup	3	80	No	No	0.27	12	14	15	19	7	8	18	22
DistalPhalanx OutlineCorrect	2	80	No	No	0.3	7	8	10	14	8	8	9	12
DistalPhalanxTW	6	80	No	No	0.27	12	14	17	26	11	13	16	25
ECG200	2	96	No	No	0.46	3	6	10	12	6	1	6	17
ElectricDevices	7	96	Partial	No	0.29	14	17	21	22	8	15	20	22
FacesUCR	14	131	Yes	No	0.55	35	37	41	61	47	56	67	93
GunPointAgeSpan	2	150	No	Yes	0.13	4	4	5	5	4	4	4	5
GunPointOld VersusYoung	2	150	No	Yes	0.12	4	5	5	6	2	2	2	2
ItalyPowerDemand	2	24	No	No	0.24	5	7	8	13	6	7	7	9
MedicalImages	10	99	No	No	0.18	5	5	7	12	7	8	9	14
MiddlePhalanx OutlineAgeGroup	3	80	No	No	0.23	10	12	14	18	10	10	13	26
MiddlePhalanx OutlineCorrect	2	80	No	No	0.24	8	9	9	10	7	8	8	9
MiddlePhalanxTW	6	80	No	No	0.25	6	6	6	9	5	5	8	9
PhalangesOutlines Correct	2	80	No	No	0.25	7	7	8	8	7	7	8	8
Plane	7	144	No	No	0.35	14	16	24	25	15	16	19	21
ProximalPhalanx OutlineAgeGroup	3	80	Partial	No	0.24	13	16	21	27	8	19	23	28
ProximalPhalanx OutlineCorrect	2	80	No	No	0.23	9	9	9	9	9	9	9	10
ProximalPhalanxTW	6	80	No	No	0.23	14	16	20	22	14	17	21	26
SmoothSubspace	3	15	No	No	0.04	2	3	6	7	2	5	6	7
SonyAIBO RobotSurface1	2	70	Yes	No	0.35	15	18	21	31	11	15	21	27
SwedishLeaf	15	128	No	Yes	0.42	20	24	36	51	18	31	40	76
TwoLeadECG	2	82	No	No	0.33	4	5	6	7	6	6	7	8
TwoPatterns	4	128	Yes	No	0.7	11	11	12	18	12	13	15	31
UMD	3	150	No	Yes	0.12	7	7	8	8	6	7	7	9



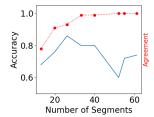


Figure 7: Accuracy (in Blue), of GPT-4.1 depending on the number of segments of the simplifications used by OS for teaching and testing. On the left ECG200 which has length 96, and on the right SonyAI-BORobotSurface1 which has length 70. In Red we see the loyalty of these simplifications.

8 Conclusion

We have introduced metrics, see Definition 1, to evaluate the use of simplification algorithms for time series in interpretability of Time Series Classifiers. We have applied these metrics to evaluate four simplification algorithms on CNN classifiers for 28 datasets from the UCR repository [3]. The results show that the OS (Optimal Simplification) and RDP (Ramer-Douglas-Peuker) algorithms have the best

performance³. The OS algorithm has time complexity cubic in the length of the original time series, and hence the cheaper RDP algorithm would be preferable for longer time series.

Our results confirm that simplifications are useful for interpretability of TSC. An important finding is that the characteristics of the datasets play a big role in how a relatively high loyalty can be achieved by a simplification on relatively few segments. For datasets that are Seasonal, Non-stationary and of Low Entropy, we get the best results, where a simplification keeping less than 5% of the original time points is sufficient to almost always preserve the classification of the original. We also performed an evaluation using multi-modal large language models and these results also support the utility of simplification algorithms in TSC interpretability.

References

- [1] E. Camponogara and L. F. Nazari. Models and algorithms for optimal piecewise-linear function approximation. *Mathematical Problems in Engineering*, 2015(1):876862, 2015.
- Note that the BU (Bottom-Up) algorithm was designed to minimize the number of segments that go over more than a single time step. However, for the purpose of the comparisons in this paper we needed to count also for BU the segments on a single time step. This explains why BU performs worse than the other algorithms. In future work we would like to compare BU to other algorithms sharing its simplification criterium.

- [2] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Meas.*, 20:37–46, 1960.
- [3] H. A. Dau, E. Keogh, K. Kamgar, C.-C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, Yanping, B. Hu, N. Begum, A. Bagnall, A. Mueen, G. Batista, and Hexagon-ML. The UCR time series classification archive, 2018.
- [4] E. Delaney, D. Greene, and M. T. Keane. Instance-based counterfactual explanations for time series classification. In *Case-Based Reasoning Research and Development - Int. Conference, ICCBR 2021*, pages 32–47. Springer, 2021.
- [5] D. Dickey and W. Fuller. Distribution of the estimators for autoregressive time series with a unit root. *JASA. Journal of the American Statistical Association*, 74, 06 1979. doi: 10.2307/2286348.
- [6] D. H. DOUGLAS and T. K. PEUCKER. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973. doi: 10.3138/FM57-6770-U75U-7727. URL https://doi.org/10.3138/ FM57-6770-U75U-7727.
- [7] A. H. Gee, D. García-Olano, J. Ghosh, and D. Paydarfar. Explaining deep classification of time-series data with learned prototypes, 2019. URL https://ceur-ws.org/Vol-2429/paper3.pdf.
- [8] Z. Geler, V. Kurbalija, M. Ivanović, and M. Radovanović. Weighted kNN and constrained elastic distances for time-series classification. Expert Systems with Applications, 162:113829, 2020.
- [9] M. Guillemé, V. Masson, L. Rozé, and A. Termier. Agnostic local explanation for time series classification. In 31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019, pages 432–439. IEEE, 2019. doi: 10.1109/ICTAI.2019.00067. URL https://doi.org/10.1109/ICTAI.2019.00067.
- [10] F. Hirschmann. rdp: Pure python implementation of the ramer-douglaspeucker algorithm. https://https://pypi.org/project/rdp/, 2025. PyPI package version 0.8.
- [11] C. Holder, D. Guijo-Rubio, and A. Bagnall. Clustering time series with k-medoids based algorithms. In *International Workshop on Advanced Analytics and Learning on Temporal Data*, pages 39–55. Springer, 2023.
- [12] B. Håvardstun, C. Ferri, and J. A. Telle. An interactive tool for interpretability of time series classification. In European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, 2024. To be published.
- [13] S. Hügel. Simplification, 12 2021. URL https://github.com/urschrei/ simplification.
- [14] A. A. Ismail, M. K. Gunady, H. C. Bravo, and S. Feizi. Benchmarking deep learning interpretability in time series predictions. In *Annual Conference on Neural Information Processing Systems* 2020, NeurIPS 2020, December 6-12, 2020, virtual, 2020.
- [15] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963, 2019.
- [16] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems*, 3:263–286, 2001.
- [17] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Proceedings 2001 IEEE International Conference on Data Mining*, pages 289–296, 2001. doi: 10.1109/ICDM. 2001.989531.
- [18] E. Keogh, S. Chu, D. Hart, and M. Pazzani. Segmenting time series: A survey and novel approach. In *Data mining in time series databases*, pages 1–21. World Scientific, 2004.
- [19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.
- [20] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
- [21] S. M. Pincus. Approximate entropy as a measure of system complexity. Proceedings of the National Academy of Sciences, 88(6):2297–2301, 1991. doi: 10.1073/pnas.88.6.2297. URL https://www.pnas.org/doi/abs/10.1073/pnas.88.6.2297.
- [22] V. Prithyani, M. Mohammed, R. Gadgil, R. Buitrago, V. Jain, and A. Chadha. On the feasibility of vision-language models for time-series classification, 2025. URL https://arxiv.org/abs/2412.17304.
- [23] A. Rajkomar, E. Oren, K. Chen, A. M. Dai, N. Hajaj, M. Hardt, P. J. Liu,

- X. Liu, J. Marcus, M. Sun, et al. Scalable and accurate deep learning with electronic health records. *NPJ digital medicine*, 1(1):1–10, 2018.
- [24] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. Computer Graphics and Image Processing, 1(3):244–256, 1972. ISSN 0146-664X. doi: https://doi.org/10.1016/S0146-664X(72)80017-0. URL https://www.sciencedirect.com/science/article/pii/S0146664X72800170.
- [25] T. Rojat, R. Puget, D. Filliat, J. Del Ser, R. Gelin, and N. Díaz-Rodríguez. Explainable artificial intelligence (xai) on timeseries data: A survey. arXiv preprint arXiv:2104.00950, 2021.
- [26] U. Schlegel, H. Arnout, M. El-Assady, D. Oelke, and D. A. Keim. To-wards a rigorous evaluation of xai methods on time series. In 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), pages 4197–4201. IEEE, 2019.
- [27] U. Schlegel, D. V. Lam, D. A. Keim, and D. Seebacher. Ts-mule: Local interpretable model-agnostic explanations for time series forecast models. 2021.
- [28] Y.-W. Si and J. Yin. Obst-based segmentation approach to financial time series. Engineering Applications of Artificial Intelligence, 26(10):2581– 2596, 2013. ISSN 0952-1976. doi: https://doi.org/10.1016/j.engappai. 2013.08.015. URL https://www.sciencedirect.com/science/article/pii/ S0952197613001723
- [29] S. A. Siddiqui, D. Mercier, M. Munir, A. Dengel, and S. Ahmed. Tsviz: Demystification of deep learning models for time-series analysis. *IEEE Access*, 7:67027–67040, 2019. doi: 10.1109/ACCESS.2019.2912823. URL https://doi.org/10.1109/ACCESS.2019.2912823.
- [30] G. A. Susto, A. Cenedese, and M. Terzi. Time-series classification methods: Review and applications to power systems data. *Big data* application in power systems, pages 179–220, 2018.
- [31] R. Tavenard, J. Faouzi, G. Vandewiele, F. Divo, G. Androz, C. Holtz, M. Payne, R. Yurchak, M. Rußwurm, K. Kolar, and E. Woods. Tslearn, a machine learning toolkit for time series data. *Journal of Machine Learning Research*, 21(118):1–6, 2020. URL http://jmlr.org/papers/v21/20-091.html.
- [32] J. A. Telle, C. Ferri, and B. A. T. Håvardstun. Optimal robust simplifications for explaining time series classifications. In *Proceedings of the Workshop on Explainable AI for Time Series and Data Streams (TempXAI 2024) co-located with (ECML-PKDD 2024)*, volume 3761 of *CEUR Workshop Proceedings*, pages 28–43, 2024. URL https://ceur-ws.org/Vol-3761/paper2.pdf.
- [33] A. Theissler, F. Spinnato, Û. Schlegel, and R. Guidotti. Explainable AI for time series classification: A review, taxonomy and research directions. *IEEE Access*, 10:100700–100724, 2022.
- [34] M. Visvalingam and J. D. Whyatt. Line generalisation by repeated elimination of points. *The cartographic journal*, 30:46–51, 1993. ISSN 0008-7041. doi: 10.1179/000870493786962263. URL https://hull-repository.worktribe.com/output/376330.
- [35] Y. Wan, X. Gong, and Y.-W. Si. Effect of segmentation on financial time series pattern matching. Applied Soft Computing, 38:346–359, 2016. ISSN 1568-4946. doi: https://doi.org/10.1016/j.asoc. 2015.10.012. URL https://www.sciencedirect.com/science/article/pii/S1568494615006341.
- [36] Wikipedia contributors. Supermajority Wikipedia, the free encyclopedia, 2025. URL https://en.wikipedia.org/wiki/Supermajority. [Online; accessed 7-May-2025].
- [37] C.-J. Wu, W.-S. Zeng, and J.-M. Ho. Optimal segmented linear regression for financial time series segmentation. In 2021 International Conference on Data Mining Workshops (ICDMW), pages 623–630. IEEE, 2021
- [38] X. Xu, H. Wang, Y. Liang, P. S. Yu, Y. Zhao, and K. Shu. Can multimodal llms perform time series anomaly detection?, 2025. URL https://arxiv.org/abs/2502.17812.

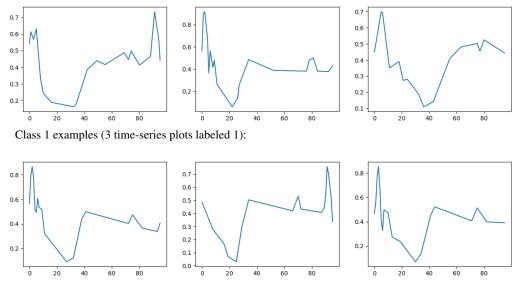
A Prompt

The following text shows an example of the prompt utilised for the experiments with LLMs mentioned in Section 7. The example is based on the ECG200 dataset. The simplifications have been generated with OS algorithm using an alpha of 0.2.

You are a time-series classification expert. Your goal is to learn from a small set of labeled examples (classes 0,1) and then assign the correct class to a new, unlabeled time series. Follow these steps:

- 1. Carefully examine the 3 examples of classes 0,1 and identify their common patterns.
- 2. Compare the new instance to your learned characteristics of classes 0,1.
- 3. Provide a brief rationale for your decision.
- 4. Conclude with one line stating only the predicted class for each one of the unlabeled examples.
- 5. Only use the pattern (Predicted class: 0,1) for each one of the unlabeled examples exclusively in the final guess.

Remember that I will always provide you with 10 unlabeled examples. Therefore, you need to perform exactly 10 predictions. The examples: Class 0 examples (3 time-series plots labeled 0):



New instances to classify (unlabeled time-series):

