Optimal Robust Simplifications for Explaining Time Series Classifications

Jan Arne Telle[1], Cèsar Ferri[2] and Brigt Håvardstun[1]

[1]Department of Informatics, University of Bergen, Norway
[2]VRAIN, Universitat Politècnica de València, Spain

Abstract

Given a Time Series Classifier (TSC) and three parameters that balance between error, simplicity and robustness, we define an optimization problem over all possible ways of simplifying a given time series ts into straight-line segments. Robustness is fixed as the fraction of perturbations that have the same classification as ts under the TSC, and we introduce a novel method for generating a set of perturbations where this information is easy to visualize. We prove that under some mild conditions on the TSC and the three parameters, we can find the optimal solution in time polynomial in the length of ts, by first doing dynamic programming to solve for error and simplicity, and then adding robustness. We test the resulting Optimal Robust Simplification (ORS)-Algorithm on binary TSCs for three datasets from UCR. We apply the ORS-Algorithm to prototypes of the classes, with varying parameters, to evaluate its power as an explanatory tool for the trained classifiers. We also provide a tool for visualizing the robustness information. We believe the resulting insights show the usefulness of the Optimal Robust Simplifications in explaining TSCs.

Keywords

Time Series, XAI, Simplification, Explainability

1. Introduction

Temporal data is encountered in many real-world applications ranging from patient data in healthcare [1] to the field of cyber security [2]. Deep learning methods have been successful [3, 1, 2] 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 [4], Saliency Methods [5] and Counterfactuals [6], and also combinations of these [7].

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 [8]. 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 [9] 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 use of this freedom, supporting the notion that time-series are particularly non-intuitive for humans.

Theissler et al [10] 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 the features important for the classification, while at the same time simplifying the instance to mitigate the non-intuitive aspects of this domain, as we attempt in this work. Theissler et al [10] 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. In this work we fill this gap, and introduce a way of *simplifying* a time series by straight-line segments, and doing this in a way that is *robust* with respect to the classification of a given TSC h.[1] Our ORS-algorithm, for Optimal Robust Simplifications, is model-agnostic, with robustness of a simplification being the fraction of local perturbations that retains the classification under TSC h. There are various ways of perturbing times series, see e.g. [11, 12, 13]. In this work we develop a different method of computing a set of perturbations, both since we start with a simplification on straight-line segments and also because we want the aggregate information about their classifications to be visualized in an informative way. When computing robustness we start with a straight-line simplification, and any perturbation must to a certain degree keep that position and shape, while it is also allowed to deviate as the perturbation moves inside a band around the simplification, see Figure 8.

The ORS-Algorithm computing a robust simplification sts of a given times series ts, under a TSC h, has three parameters that allows to balance between error (Squared Euclidean distance between ts and sts), simplicity (number of segments of sts), and robustness (what fraction of perturbations of sts have same classification as ts by h), and sts is defined as the minimization over an objective function taking all this into account, see Definition 1. Perhaps surprisingly, in Theorem 4 we show that under some mild conditions on h and the three parameters, we can find the optimal solution in time polynomial in the length of the original time series ts, by first solving for error and simplicity, and then adding robustness. Note that our algorithm allows not only the balance between error, simplicity and robustness to change, but also their computation, i.e. the algorithm is agnostic to the particular methods used to calculate these values. Hence Squared Euclidean distance can be replaced by some other distance function as long as it is amenable to the dynamic programming, simplicity can depend on other aspects than number of segments, and robustness can be computed by other types of perturbations.

In the rest of this paper we first discuss related work and cover some standard definitions before presenting the ORS-Algorithm together with proof of correctness and runtime. We then discuss the usefulness of the robust simplifications, with a focus in this paper on time series that are discrete, univariate, evenly sampled, aperiodic, short, and non-stationary, based on UCR datasets Chinatown, Italy Power Demand and ECG200. We also provide an aid to visualize the robustness information. Our findings suggest that the robust simplifications of a prototype ts can be used to extract explanations of the classification done by the TSC, as they simplify ts to the point where human intuition can more easily be applied. We conclude the paper by discussing some directions for future work.

2. Related Work

Several techniques have been applied to generate explanations from TSC models [10, 14]. Specifically, techniques previously used on Convolutional Neural Networks or Recurrent Neural Networks have been applied when these techniques have been used for TSC. For instance, the authors in [7] 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. Related to the datasets used in our paper, [15] presents a user study of XAI methods to explain the ECG200 dataset from UCR.

Another alternative is to produce explanations through examples, and these can be specifically utilized 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 behavior of a similar sample [16]. A method to generate prototypes for time series data using an autoencoder is presented in [17]. 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 [18], the authors propose a dimensionality reduction technique for time series, called Piecewise Aggregate Approximation, as a tool for similarity search

[1] Note that if one pieces together several such local explanations, say for prototypes of each class, then this can still form a global explanation of the given TSC h.

in large time series databases. The work [19] presents a review of piecewise linear approximations employed for time series data, and the authors also propose a method named SWAB (based on the Sliding Window and Bottom-up approaches). Finally, in [20] 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 [21] 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 [22]. In [23], Si and Yin also apply segmentation to financial time series data, now as a preprocessing step to locate technical patterns.

Perturbations have also been previously employed for XAI and time series. In [11], the authors propose a framework to generate explanations for time series classifications based on a generative model to create with-in-distribution perturbations. Enguehard, in [12], introduces a technique to explain multivariate time series predictions using a perturbation-based saliency method. Recently, the approach ContraLSP has been detailed in [13]. This method is based on a locally sparse model that produces counterfactual samples to build uninformative perturbations but keeps distribution using contrastive learning.

Some tools have been proposed to allow users to interact with time series and, in that way, to get some insights about their classification, like [24] where model inspection is the key aspect. In [25], the authors develop an interactive XAI tool for loan applications that allows users to experiment with hypothetical input values and inspect their effect on the outcomes of the model and perform a user evaluation on MTurk. [26] presents a Python package to provide a unified interface for the interpretation of time series classification.

Some papers have developed a similar approach to our proposal. Im [27], Tang et al. presents the method Dual Prototypical Shapelet Networks (DPSN) for few-shot time-series classification. This method trains a neural network from few examples and also interprets the model from two levels of granularity: a global overview with representative time series examples, and local highlights with discriminative shapelets. Another related paper for sequence learning is [28]. The work proposes ProSeNet a RNN-based method for deep sequence modeling. The approach combines prototype learning and RNNs to construct models with high accuracy and interpretability. The method considers three criteria in constructing prototypes: Simplicity, the prototypes can be subsequences with only the key events determining the output; Diversity, redundant prototypes should be avoided; Sparsity, for each example to explain only a few prototypes are shown to avoid long explanations. The authors show the validity of ProSeNet for time series using the MIT-BIH Arrhythmia ECG dataset.

3. Standard definitions

Let us present formal definitions for Time Series Classification (TSC) and recall basic notions.

Staying consistent with earlier notation [10, 6] a time series $T = \{t_1, t_2, \ldots, t_m\}$ is an ordered set of m real-valued observations (or time steps). Note we may also view each t_i as a pair consisting of an x-value (the time) and a y-value (the observation). A time series dataset $D = \{T_1, T_2, ..., T_n\} \in R^{n \times m}$ is a collection of such time series where each time series has a class label c forming a vector of class labels. In this paper we consider only binary classification tasks. Given such a dataset, Time Series Classification is the task of training a mapping b from the space of possible inputs to a probability distribution over the class values. Thus, a black-box classifier $b(T)$ takes a time series T as input and predicts a probability output over the class values.

Prototypes are time series exemplifying the main aspects responsible for a classifier's specific decision outcome. It can be a real instance (which is what we opt for) sampled from the dataset that is important and meaningful because it summarizes the shape of many other similar instances, or a synthetic one, for example a cluster centroid or an instance generated by following some ad-hoc processes.

4. An Algorithm for Optimal Robust Simplifications

In this section we give the ORS-Algorithm that takes a binary Time Series Classifier h and a univariate times series ts, on n points $p_1, ..., p_n$ given by increasing x-values, and computes the optimal robust simplification $opt_{simp}(ts, h, \alpha, \beta, \gamma)$, where α, β, γ are factors freely chosen to balance between error, simplicity and robustness. This simplification will select $k+1$ of the n points $p_{i_1}, p_{i_2}, ..., p_{i_{k+1}}$ with $i_j < i_{j+1}, \forall j$ to form a simplified time series sts on k segments by drawing a line between each consecutive pairs of the $k+1$ points and by letting the first segment and the last segment extend to the x-values of p_1 and p_n, respectively. See Figure 1.

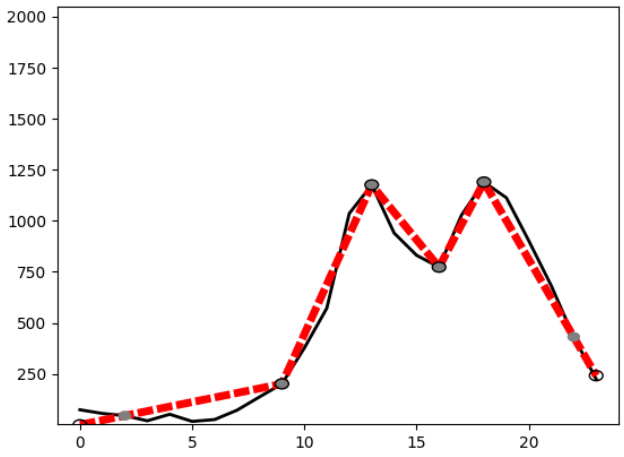


Figure 1: Original time series ts in black, with selected points given by 6 grey circles, resulting in a simplification sts in red with 5 segments.

Definition 1. *We define $opt_{simp}(ts, h, \alpha, \beta, \gamma)$ formally as the minimum, over all $2^n - n - 1$ possible choices of $2 \leq k \leq n$ points that give a simplification sts with $h(ts) = h(sts)$, of the value*

$$\alpha \cdot err(ts, sts) + \beta \cdot k_{sts} + \gamma \cdot frag(ts, sts, h)$$

comparing the original time series ts and the simplification sts on $k_{sts} = k - 1$ segments defined by these k points.

The optimal simplification thus reflects a selected balance of error, simplicity and robustness achieved by freely choosing the 3 factors α, β, γ:

- α for error: $err(ts, sts)$ is Squared Euclidean distance between ts and sts, i.e. the sum over all x-values, of the square of the difference between the corresponding y-value of ts and sts.
- β for simplicity: k_{sts} is the number of segments
- γ for robustness: $frag(ts, sts, h)$ is a measure of how robust the classifier h is on perturbations of sts, measured by the fraction of perturbations that do not fall in the same class as ts, thus in the range $[0, 1]$ with 0 being most robust ($frag$ is short for fragility and can be viewed as (1-robustness))

Perhaps surprisingly, under some mild assumptions this complicated objective function still allows an algorithm that finds the optimal in time polynomial in n. In this section we describe this algorithm in detail.

4.1. If no robustness: Dynamic Programming

We first solve the case of $\gamma = 0$, i.e. without giving any weight to robustness. Firstly, Least Squares is a foundational problem in statistics and numerical analysis, given points $p_1, p_2, ..., p_n$ in the plane find the straight line that minimizes the squared error. In the more general Segmented Least Squares (SLS) problem we ask for a sequence of straight-line segments that minimizes cost, which is some balance of error and simplicity (number of segments). The SLS problem is famously solvable in polynomial time by a textbook Dynamic Programming Algorithm based on the following recurrence for $OPT(j)$, the minimum cost for points $p_1, ..., p_j$:

$$OPT(j) = \begin{cases} 0, & \text{if } j = 0 \\ \min_{1 \le i \le j}\{lse(i,j) + \beta + OPT(i-1)\} & \text{else} \end{cases}$$

where $lse(i, j)$ is the least square error over all single lines that cover p_i to p_j, and β is the the cost of one extra segment, with value of β chosen to balance between error and simplicity. This problem formulation differs from our setting in 3 important ways:

- We want a continuous set of segments, with each segment starting and ending in given points.
- We want the first (resp. last) segment to be defined by any two points p_i, p_j and the line drawn between them continued until it hits the x-value of p_1 (resp. of p_n).
- We want robustness to perturbations as part of the objective function.

The first two differences are relatively easy to accommodate, as follows:

$$OPT(j) = \begin{cases} 0, & \text{if } j = 0 \\ \min_{1 \le i \le j}\{[\alpha \cdot err(i,j) + \beta + OPT(i)], [\alpha \cdot err(1,i,j) + \beta]\} & \text{if } j < n \\ \min_{1 \le i \le n} \min_{i+1 \le k \le n}\{[\alpha \cdot err(i,k,n) + \beta + OPT(i)], & \\ \qquad [\alpha \cdot err(1,i,k,n) + \beta]\} & \text{if } j = n \end{cases}$$

where

- $err(i, j)$ is the Squared Euclidean distance between the straight line from p_i to p_j and the part of the original time series ts given by points $p_i, ..., p_j$
- $err(i, j, n)$ is the same except the straight line continues past p_j to the x-value of p_n and we compare to the final part of ts given by $p_i, ..., p_n$
- $err(1, i, j)$ is similar as above except the line continues in the other direction to the x-value of p_1 and we compare to the initial part of ts $p_1, ..., p_j$
- $err(1, i, j, n)$ compares ts to a single line covering all x-values and going through points p_i, p_j

For the case of $opt_{simp}(ts, h, \alpha, \beta, \gamma)$ where $\gamma = 0$ i.e. where robustness does not play a part, the argument for correctness of the above recurrence is similar to the textbook argument for correctness of the recurrence of SLS, so we do not repeat it here, except to note the new parts being that

- We use $OPT(i)$ rather than $OPT(i-1)$ since now for two consecutive segments the endpoint of the first is the startpoint of the next.
- When defining $OPT(j)$ we use $err(1, i, j)$ so first segment can end in j
- When defining $OPT(n)$ we use $err(i, k, n)$ so last segment can choose any pair p_i, p_k, and $err(1, i, k, n)$ to allow simplification with a single segment.

Transforming this recurrence into a dynamic programming algorithm is straightforward, by first computing each of the $O(n^2)$ error terms in time $O(n)$ each, followed by a nested loop computing the n $OPT(j)$ values in $O(n)$ time each. Retrieving the optimal solution is done by a second pass over the values stored, also standard. This gives the following result.

Theorem 1. *Given a time series ts of length n, a binary TSC h, and factors α, β, γ that balance between error, simplicity and robustness. For the case of $\gamma = 0$ (no robustness) we can compute the optimal simplification achieving the minimum $opt_{simp}(ts, h, \alpha, \beta, \gamma = 0)$ in time $O(n^3)$ by dynamic programming.*

4.2. Accommodating Robustness: DP with Heaps

Dynamic programming relies on the property that an optimal solution to a larger problem consists of optimal solutions to subproblems. Robustness depends on the TS Classifier h and we cannot expect it to satisfy this property, since its classification on only a part of a time series will not in general correspond to its classification on the full time series. Thus, we cannot incorporate robustness as part of the DP scheme. Instead, our algorithm for Optimal Robust Simplifications (ORS), the **ORS-Algorithm**, uses a 2-stage approach:

Stage 1 Use DP to compute a 2-dimensional table of size $n \times q$ that in cell $D[j,k]$ stores the kth least costly simplification for the points $p_1, ..., p_j$ of the input time series ts, considering only error and simplicity but not robustness, as in above DP.

Stage 2 Consider the q least costly simplifications $sts_1, ..., sts_q$, ordered by non-decreasing cost under error and simplicity as stored in $D[n,1]..D[n,q]$ respectively. Compute robustness for any simplification that h classifies same as ts.
The ORS-Algorithm then returns the simplification having lowest overall total cost, i.e. the sts minimizing the value of value of $\alpha \cdot err(ts, sts) + \beta \cdot k_{sts} + \gamma \cdot frag(ts, sts, h)$

Pseudo-code is given in the Appendix. We will here give a high-level explanation, but let us first state the following formal result.

Theorem 2. *Let the difference in cost between sts_1 and sts_q, as computed by Stage 1, be d. For any value of $\gamma \leq d$ the simplification returned by the ORS-Algorithm will then be an optimal one achieving $opt_{simp}(ts, h, \alpha, \beta, \gamma)$.*

Proof 1. *We prove the statement by contradiction. Let the simplification sts returned by the ORS-Algorithm have k_{sts} segments and assume there is another simplification sts' with $k_{sts'}$ segments such that*

$$\alpha \cdot err(ts, sts') + \beta \cdot k_{sts'} + \gamma \cdot frag(ts, sts', h) <$$

$$\alpha \cdot err(ts, sts) + \beta \cdot k_{sts} + \gamma \cdot frag(ts, sts, h)$$

Since Stage 2 optimized this value over all simplifications $sts_1, ..., sts_q$ we cannot have sts' among these, and thus by the assumption in the statement of the theorem we have $\alpha \cdot err(ts, sts') + \beta \cdot k_{sts'} \geq d + \alpha \cdot err(ts, sts_1) + \beta \cdot k_{sts_1}$, with k_{sts_1} the number of segments of sts_1. Thus we get

$$d + \alpha \cdot err(ts, sts_1) + \beta \cdot k_{sts_1} + \gamma \cdot frag(ts, sts', h) <$$

$$\alpha \cdot err(ts, sts) + \beta \cdot k_{sts} + \gamma \cdot frag(ts, sts, h) \leq$$

$$\alpha \cdot err(ts, sts_1) + \beta \cdot k_{sts_1} + \gamma \cdot frag(ts, sts_1, h)$$

with the latter inequality following since sts was the minimum over $sts_1, ..., sts_q$. Rearranging, this gives $d < \gamma \cdot frag(ts, sts_1, h) - \gamma \cdot frag(ts, sts', h) + ((\alpha \cdot err(ts, sts_1) + \beta \cdot k_{sts_1}) - (\alpha \cdot err(ts, sts_1) + \beta \cdot k_{sts_1}))$ with the latter term being zero and thus

$$d < \gamma \cdot (frag(ts, sts, h) - frag(ts, sts', h))$$

Note we have $frag(\cdot)$ in the range $[0, 1]$ which means the above implies that $\gamma > d$, contradicting the assumption in the statement of the theorem. □

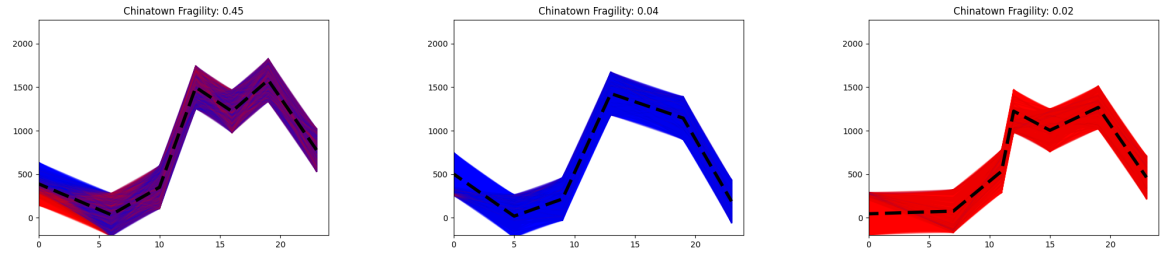
4.2.1. Important details of Stage 1.

We describe some key details of how to implement Stage 1, computing a table of size $n \times q$ with $D[j,k]$ storing the kth least costly simplification of points $p_1, ..., p_j$ under error and simplicity only. The difference to the earlier DP is that we compute not only the least costly but the q least costly with $q > 1$. After computing error terms we run an outer loop from $j = 1$ to n with two inner loops one after the other. The first loop from $i = 1$ to $j - 1$ is similar to the previous DP, computing the cost $D[i,1] + err(i,j)$ of having the last segment go from p_i to p_j, but now adding all these values to a heap H_j. The second loop from $k = 1$ to q fills the $D[j,k]$ entries by extracting the minimum element from H_j, say this element is $D[i,r] + err(i,j)$, then first set $D[j,k]$ to this element, but most importantly also add to the heap H_j the new value $D[i,r+1] + err(i,j)$ since the $r+1$st least costly solution up to p_i may be lower than many other solutions in the heap. For details, in particular regarding the edge-cases which are cumbersome to implement properly, see the pseudo-code in the appendix.

4.2.2. Computing Robustness in Stage 2.

A simplified time series sts on k segments is computed by choosing $k+1$ points $p_{i_1}, ..., p_{i_{k+1}}$ from an original time series ts on points $p_1, ..., p_n$. When computing the robustness of sts focus on the following $k+1$ *pivot points* $s_1, p_{i_2}, ..., p_{i_k}, s_2$ of sts (note the first and last may not be points of ts) where s_1 (and s_2) is the y-value of sts that corresponds to the smallest (and largest) x-value, where smallest is same as x-value of p_1 (and largest is same as x-value of p_n). In Figure 1 s_1 is the leftmost circled point and s_2 the rightmost circled point, at x-values 0 and 23 respectively. For these $k+1$ pivot points of sts we allow an increase or decrease of its y-value by an ϵ which is 10% of the y-range of the dataset the classifier h was trained on, i.e. $\epsilon = 0.1 * (y_{max} - y_{min})$ with y_{max} and y_{min} the maximum and minimum y-value in this dataset.

We compute 10.000 random perturbations of sts, each consisting of k segments anchored at the same x-values as the $k+1$ pivot points of sts but with the y-value of each pivot point shifted by an amount drawn independently for the $k+1$ values uniformly at random in the range $[-\epsilon, +\epsilon]$. Thus the perturbation will lie inside a band around sts of height $2 \cdot \epsilon$ at each pivot point. See Figure 2. We then use h to classify each of the perturbations, and define the $frag$-value of sts as the fraction of the 10.000 random perturbations that have the opposite classification as ts.



(a) $frag = 0.45$ blue. Not robust.

(b) $frag = 0.04$ blue. Very robust.

(c) $frag = 0.02$ red. Very robust.

Figure 2: Visualizing robustness of the three simplified time series in dotted black lines. All perturbations consist of straight-line segments that are forced to lie inside the colored band, with pivot points at same x-values as the simplification, but different y-values. We compute several perturbations of sts by randomly altering the y-values of the $k_{sts}+1$ pivot points inside the narrow band, e.g. in (b) sts in black has 5 segments and the 6 pivot points of the perturbations are at x-values 0,5,9,13,18,24. Each perturbation is drawn with a color according to its classification by the classifier h. Thus if many have same color as ts then robustness is good (and $frag$-value low).

Theorem 3. *The runtime of the ORS-ALgorithm is* $O(n \cdot \max\{n^2, q \log n\})$.

Proof 2. *The initialization of error terms is $O(n^3)$ as in the first DP version. We use binary heaps having $O(\log n)$ insertion and extraction operations. Then, for each of the n values of $1 \leq j \leq n$ we have two for-loops, one inserting at most n values into heap H_j (for time $O(n \log n)$), and the other extracting and inserting at most q values of H_j (for time $O(q \log n)$). Note there are never more than n elements in any heap H_j as the first loop never inserts more than n elements and the latter loop extracts one element and inserts at most one new element. This completes the proof.*

Theorem 4. *The optimal simplification $opt_{simp}(ts, h, \alpha, \beta, \gamma)$ can be computed in polynomial time for a time series ts on n points, a binary TSC h, and balancing factors α, β, γ, if there exists a constant c, such that the difference is $\geq \gamma$ between the least costly simplification that does not take robustness into account and the n^cth least costly one.*

Proof 3. *This follows from Theorems 2 and 3 by running the ORS Algorithm with $q = n^c$.*

5. Usefulness of the robust simplifications

We have implemented [2] the ORS-Algorithm and in this section we illustrate its use for explaining a Time Series Classifier. The algorithm is designed for classifiers working on univariate discrete time series with a binary classification. Moreover, we believe its use is more easily evaluated if the times series are evenly sampled, aperiodic and non-stationary. However, it is important to note that apart from the above constraints we did not want simple datasets where the binary classification is particularly easy or could be described (e.g. by ourselves) in any straightforward way. Based on these criteria we have chosen to focus on three datasets.

- From UCR [29]: Chinatown. This dataset shows the number of pedestrians on a street corner of Chinatown in Melbourne over a 24-hour time period, and classifies these into Weekend and Weekdays.
- From UCR [29]: ECG200. This dataset traces the electrical activity recorded in 96 time steps during one heartbeat and classifies the time series into a normal heartbeat and a Myocardial Infarction.
- From UCR [29]: Italy Power Demand. This dataset shows power demand in Italian households over a 24-hour time period, and classifies these into Winter (October-March) and Summer (April-September).

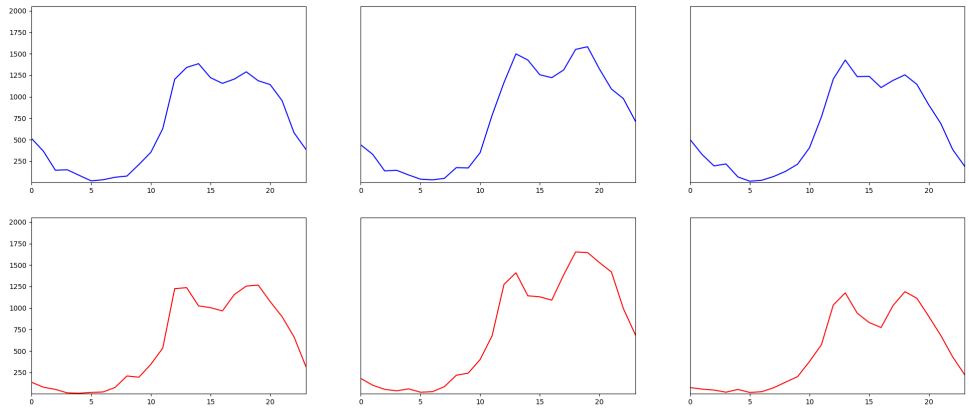


Figure 3: Six prototypes from Chinatown. Hard to make a rule-of-thumb.

[2] Available on Github: https://github.com/BrigtHaavardstun/kSimplification

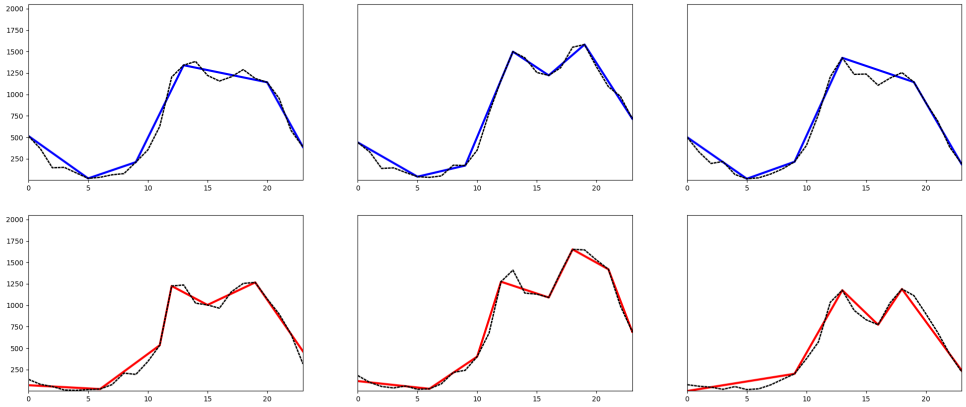


Figure 4: Same six prototypes from Chinatown, now shown in dotted black lines, with robust simplifications making it easier to extract a rule-of-thumb.

For each of these datasets we trained a fully convolutional neural network (FCN), originally proposed by Wang et al. [30]. Specifically we implemented a model closely following the work by Delaney et al. [6]. To select prototypes we used SPOTGreedy by Gurumoorthy et al. [31], implemented in the python package InterpretML[32]. We then ran the robust simplification algorithm on the resulting prototypes, with varying balancing factors, to evaluate their power as explanatory tools for the trained classifiers. In this section we focus on three aspects of the robust simplifications to argue that they are helpful explanations of the classifier model.

- Showing prototypes together with their simplifications is more informative than showing prototypes only
- Varying the balance of error vs simplicity vs robustness will highlight different aspects of the prototypes
- Extracting a rule for class membership from this information can give simple and powerful explanations

We cover these aspects in separate subsections.

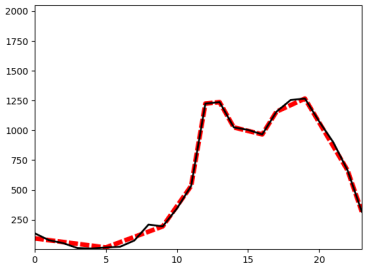
5.1. Simplifications are informative as explanations

5.1.1. Chinatown

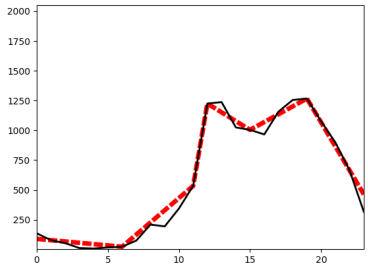
For the Chinatown dataset we have computed 3 prototypes from each of the two classes, that we call Red and Blue. These 6 prototypes are presented in Figure 3. From these prototypes only it seems difficult to identify with any kind of certainty a rule-of-thumb that can be used to decide the classification of new instances. On the next figure, Figure 4, we show a single simplification added to each of the prototypes. These simplifications are chosen with a well-balanced mix of error, simplicity and robustness, choosing each of α, β, γ in the mid-range of values [3] These simplifications function as explanations for the classification of each of the prototypes, as they also incorporate the robustness to 10.000 perturbations, and as such they allow a rule-of-thumb to be more readily guessed at. Looking at the first segment of each of the robust simplifications it is striking that the blue slopes are quite sharply downwards, whereas none of the red are. A natural guess is as follows:

- Rule-of-thumb for Chinatown classification: if the time series starts by a noticable downward slope then it is Blue, otherwise Red

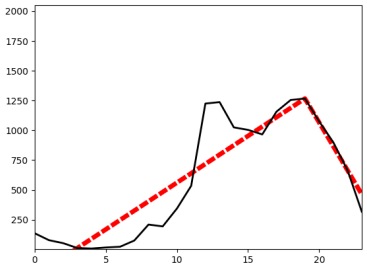
[3] Exact values used for parameters can be found on the github page. Here we only refer to them as Low-Mid-High to keep the presentation simple.



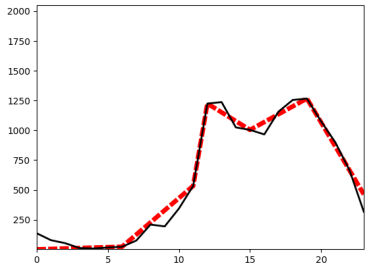
(a) 11 segs: Mid α - Low β - Mid γ



(b) 6 segs: Mid α - Mid β - Mid γ

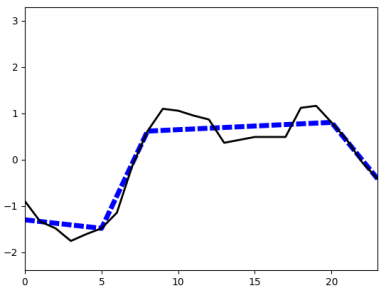


(c) 2 segs: Mid α - High β - Mid γ

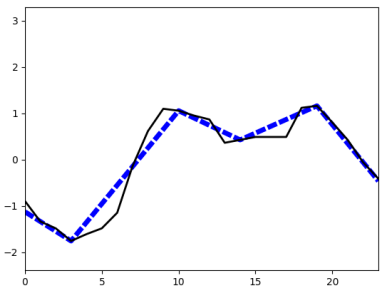


(d) 6 segs: Mid α - Mid β - High γ

Figure 5: Prototype from Chinatown in black and four distinct simplifications of it in red. From (a) to (b) to (c) we see that increasing the importance of simplicity from low to middle to high will decrease the number of segments from 11 to 6 to 2, and increase the Euclidean distance. In (d) we see that increasing the importance of robustness, when comparing to (b) which also has 6 segments, changes the slope of the first segment even though this increases the Euclidean distance, which is significant information about the red classification.



(a) 4 segs: Mid α - Mid β - Mid γ



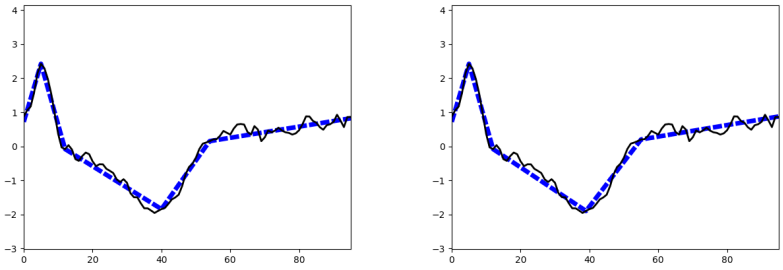
(b) 5 segs: Mid α- Mid β - High γ

Figure 6: Prototype from ItalyPowerDemand in black and two distinct simplifications of it in blue. Note that both have the same balancing factor for error and simplicity, but (a) has 4 segments while (b) has a higher factor of robustness that yields 5 segments. Simplifying, we could say that a segment in (a) has been replaced by two segments in (b) to give two peaks. This indicates that those two peaks at or around those x-values is important for the robustness of the Blue classification.

5.2. Varying balancing factors

5.2.1. Chinatown

In Figure 5 we see the effect of varying the balancing factors for a Red prototype in the Chinatown dataset. We see that increasing the importance of simplicity from low to middle to high will decrease the number of segments, while naturally also increasing the Euclidean distance to the original time series. We also see that increasing the importance of robustness changes the slope of the first segment from slightly downward (in the other 6-segment simplification) to slightly upward. This means that



(a) 5 segs: Mid α - Mid β - Mid γ **(b)** 5 segs: Mid α - Mid β - High γ

Figure 7: Prototype from ECG200 in black and two distinct simplifications of it in blue. Note that even though the factor for robustness is much higher in (b) than (a), the two simplifications are almost identical (difference being that in (a) the third segment ends in $x = 40$ whereas in (b) it ends in $x = 39$).

a higher percentage of time series with first segment having slightly downward slope in this vicinity are classified as Blue, as compared to the percentage classified Blue having slightly upward slope. This change is significant information, and it holds even in the presence of a higher Euclidean distance to the original prototype. Note that this observation constitutes supporting evidence for the rule-of-thumb guessed at previously for Chinatown.

5.2.2. Italy Power Demand

In Figure 6 we see a single prototype from ItalyPowerDemand in black and two robust simplifications of it in blue. Both simplifications have the same balancing factor for error and simplicity, but (a) has 4 segments while (b) has 5 segments resulting from an increase in the factor for robustness. Simplifying, we could say that a single segment in (a) has been replaced by two segments in (b) to give two peaks. This indicates a rule-of-thumb for the classifier trained on ItalyPowerDemand, namely that two such peaks at or around those x-values is important for the Blue classification.

5.2.3. ECG200

In Figure 7 we see a prototype from ECG200 in black and two distinct simplifications of it in blue. Note that even though the factor for robustness is much higher in (b) than (a), the two simplifications are (almost) identical, both on 5 segments. To explain this lack of significance of robustness we turn to Theorem 2 which says that the difference between the best cost and qth best cost output by Stage 1 of the ORS Algorithm, may limit the values of γ for which Stage 2 returns the optimal robust simplification. For the simplifications of Figure 7 we ran Stage 1 with the high value of $q = 1.000.000$ and found this difference to be 0.03630-0.02884=0.00746. Thus by Theorem 2 the simplification in (a) may not be optimal for that value of $\gamma = 0.01 > 0.00746$, whereas the one in (b) is likely not optimal, as $\gamma = 0.1 >> 0.00746$. Let us explain why we believe this happens. This time series has a length of 96, and note the DP to get 5 segments optimizes over all possible ways of choosing 6 points from 96. Note that there are many such choices of 6 points that give 5 segments similar to the ones in Figure 7. Counting the number of ways of choosing these 6 points, starting with the rightmost point and going left, a back-of-the-envelope calculation estimates this to be in the ballpark of 35*20*10*10*3*5 which is over 1 million. Thus, we take this as evidence that almost all the q best simplifications returned by the dynamic programming of Stage 1, that do not take robustness into account, are very close to these 5 segments. Thus increasing the importance of robustness cannot help, as the result will always be a simplification similar to the one given. We leave for future work to deal with such cases, for example by a heuristic during the DP that discards simplifications that are only marginally different from others.

5.3. Evaluating a classification rule guessed from robust simplifications

5.3.1. Chinatown

For the Chinatown classification we made a rule-of-thumb based on information gathered from the explanations of the classifications given for 6 prototypes, by means of the robust simplifications. To check if this rule-of-thumb is actually useful we need to make it more specific. Looking at the values on the y-axis for Figure 4 we see that the first segments of the 3 Blue simplifications are lines starting at point $(0, 500)$ and decreasing to about $(5, 0)$ while the first segments of the Red simplifications are only very slightly decreasing or increasing. Thus, the difference between the y-values at $x = 0$ and $x = 5$ would distinguish between these Red and Blue prototypes. We guess that this difference also distinguishes between many other time series in the dataset, as this insight is gathered from the robust simplifications. Taking the halfway point of 250 between the y-values of 500 and 0 as the cutoff the simple classification rule becomes

- Simple Rule for Chinatown: if the y-value at $x = 0$ minus the y-value at $x = 5$ is larger than 250 then ts is classified Blue, otherwise Red

The Chinatown dataset has 363 time series. We checked their classification by the model and compared to the Simple Rule. Indeed, the Simple Rule classifies all but 1 of the model-classified Blue time series as Blue, and it classifies all but 19 of the Red time series as Red, for an accuracy of 94.5 $\%$. Note this is accuracy with respect to the model, not the ground truth, as the Simple Rule is trying to explain the model. Let us mention that the accuracy of the model wrt ground truth is 96.4 %, while accuracy of the simple rule wrt ground truth is 97 %.

Finally, let us also mention that if we raise the cutoff in the Simple Rule from 250 to 325 it actually achieves an accuracy of 99.7 % (with accuracy of the simple rule vs ground truth down to 96.7%) failing on only one instance. In retrospect, we could ask if there was any information available to us from the robust simplifications that would point to this higher cutoff. In fact, consider Figure 5 (a) that visualizes the robustness of a simplification whose first segment goes from $(0, 400)$ to $(7, 50)$. Note the red part of the band around the first segment stretches up above $(0, 250)$. This should have made us wonder if the cutoff for the simple rule should be higher than 250. We could have made a test of this, for example by constructing a time series whose first segment goes from $(0, 300)$ to $(5, 0)$ and visualize its robustness. See Figure 8, where we have done exactly this, and pay close attention to the colourings of the band around the first segment. This band is predominantly blue down to somewhere above $(0, 300)$, and already then it turns red. This in itself could have made us guess a higher cutoff than 250 for the Simple Rule. We believe these insights show the usefulness of the robust simplifications and their visualization.

5.3.2. ItalyPowerDemand

In the previous subsection we saw that two peaks in the mid-afternoon seemed to signify Blue class for the classifier on the ItalyPowerDemand dataset. Looking more closely at Figure 6 we note that the difference between the high part of the peaks and the low part is about 0.5. We also note from the original prototype that the first peak is centered at 10, the second peak at 18, and that in (b) the bottom of the simplification is centered at 14. From this we extract the following rule:

- Simple Rule for ItalyPowerDemand: let $max1$ = maximum of the y-values over $x = 9, 10, 11$, let $min2$= minimum of the y-values over $x = 13, 14, 15$ and $max3$ = maximum of the y-values over $x = 17, 18, 19$. If $max1 - min2 \geq 0.5$ and $max3 - min2 \geq 0.5$ then classify this instance Blue, else Red.

The dataset ItalyPowerDemand has 1096 time series. We checked their classification by the model and compared to the Simple Rule. Indeed, the Simple Rule classifies all but 50 of the model-classified Blue correctly, and all but 49 of the Red correctly, for an accuracy of 91%. Let us mention that accuracy of the model wrt ground truth is 96.3%, and of the Simple Rule wrt Ground Truth 89.6%. Note that

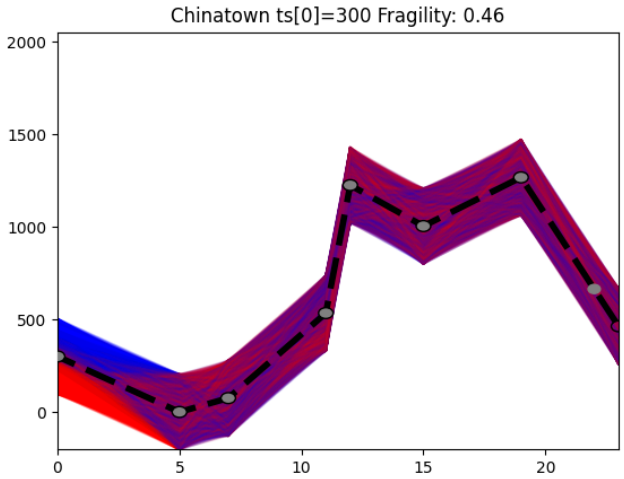


Figure 8: Visualization of robustness of a time series starting at $(0, 300)$ suggesting that a cutoff somewhat higher than $(0, 250)$ for Blue vs Red would have been a better guess for the Simple Rule for Chinatown.

classifiers of ItalyPowerDemand have previously been shown hard to explain to end-users [24]. Thus, the Simple Rule, developed by examining the robust simplifications in Figure 6, has a surprisingly high accuracy.

6. Conclusions and Future Work

In this paper, we addressed the challenge of interpretability in Time Series Classification (TSC) by introducing the Optimal Robust Simplifications (ORS) algorithm. Our approach focuses on simplifying time series data into straight-line segments while ensuring that these simplifications remain robust with respect to the classifications made by a given TSC model.

Our ORS-Algorithm is model-agnostic and allows a balance between error (fidelity with respect to the original instance), simplicity (complexity of the simplification), and robustness (fraction of local perturbations that retain the correct classification). We prove that the ORS-Algorithm, under certain mild conditions, finds the optimal solution in polynomial time, by a first dynamic programming stage solving for error and simplicity, and then incorporating robustness into the solution. This simplification process aids in making time series data more intuitive and interpretable for humans.

Our experimental results, based on UCR datasets Chinatown, Italy Power Demand, and ECG200, confirm the practical utility of our approach. The robust simplifications provided by the ORS algorithm make time series data more accessible to human intuition, facilitating better user understanding and trust in the TSC models. In conclusion, our work contributes a novel method for enhancing the interpretability of TSC models through robust simplifications.

Future research could explore other ways to compute the factors employed in the algorithm. For instance, the squared Euclidean distance could be replaced by some other distance function, or simplicity can depend on other aspects than only the number of segments, and robustness can be computed by other types of perturbations. Finally, we plan to conduct experiments including human studies to demonstrate empirically the benefits of using optimal robust simplifications to explain TSC models and compare to related XAI methods.

References

[1] 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

(2018) 1–10.
[2] G. A. Susto, A. Cenedese, M. Terzi, Time-series classification methods: Review and applications to power systems data, Big data application in power systems (2018) 179–220.
[3] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, P.-A. Muller, Deep learning for time series classification: a review, Data mining and knowledge discovery 33 (2019) 917–963.
[4] M. Guillemé, V. Masson, L. Rozé, 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, IEEE, 2019, pp. 432–439. URL: https://doi.org/10.1109/ICTAI.2019.00067. doi:10.1109/ICTAI.2019.00067.
[5] A. A. Ismail, M. K. Gunady, H. C. Bravo, 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.
[6] E. Delaney, D. Greene, M. T. Keane, Instance-based counterfactual explanations for time series classification, in: Case-Based Reasoning Research and Development - Int. Conference, ICCBR 2021, Springer, 2021, pp. 32–47.
[7] U. Schlegel, H. Arnout, M. El-Assady, D. Oelke, D. A. Keim, Towards a rigorous evaluation of xai methods on time series, in: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), IEEE, 2019, pp. 4197–4201.
[8] S. A. Siddiqui, D. Mercier, M. Munir, A. Dengel, S. Ahmed, Tsviz: Demystification of deep learning models for time-series analysis, IEEE Access 7 (2019) 67027–67040. URL: https://doi.org/10.1109/ACCESS.2019.2912823. doi:10.1109/ACCESS.2019.2912823.
[9] B. Håvardstun, C. Ferri, 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.
[10] A. Theissler, F. Spinnato, U. Schlegel, R. Guidotti, Explainable AI for time series classification: A review, taxonomy and research directions, IEEE Access 10 (2022) 100700–100724.
[11] H. Meng, C. Wagner, I. Triguero, Explaining time series classifiers through meaningful perturbation and optimisation, Information Sciences 645 (2023) 119334. URL: https://www.sciencedirect.com/science/article/pii/S0020025523009192. doi:https://doi.org/10.1016/j.ins.2023.119334.
[12] J. Enguehard, Learning perturbations to explain time series predictions, in: International Conference on Machine Learning, PMLR, 2023, pp. 9329–9342.
[13] Z. Liu, Y. ZHANG, T. Wang, Z. Wang, D. Luo, M. Du, M. Wu, Y. Wang, C. Chen, L. Fan, Q. Wen, Explaining time series via contrastive and locally sparse perturbations, in: The Twelfth International Conference on Learning Representations, 2024. URL: https://openreview.net/forum?id=qDdSRaOiyb.
[14] T. Rojat, R. Puget, D. Filliat, J. Del Ser, R. Gelin, N. Díaz-Rodríguez, Explainable artificial intelligence (xai) on timeseries data: A survey, arXiv preprint arXiv:2104.00950 (2021).
[15] C. Obermair, A. Fuchs, F. Pernkopf, L. Felsberger, A. Apollonio, D. Wollmann, Example or prototype? learning concept-based explanations in time-series, in: Asian Conference on Machine Learning, PMLR, 2023, pp. 816–831.
[16] Z. Geler, V. Kurbalija, M. Ivanović, M. Radovanović, Weighted kNN and constrained elastic distances for time-series classification, Expert Systems with Applications 162 (2020) 113829.
[17] A. H. Gee, D. García-Olano, J. Ghosh, D. Paydarfar, Explaining deep classification of time-series data with learned prototypes, 2019. URL: https://ceur-ws.org/Vol-2429/paper3.pdf.
[18] E. Keogh, K. Chakrabarti, M. Pazzani, S. Mehrotra, Dimensionality reduction for fast similarity search in large time series databases, Knowledge and information Systems 3 (2001) 263–286.
[19] E. Keogh, S. Chu, D. Hart, M. Pazzani, Segmenting time series: A survey and novel approach, in: Data mining in time series databases, World Scientific, 2004, pp. 1–21.
[20] E. Camponogara, L. F. Nazari, Models and algorithms for optimal piecewise-linear function approximation, Mathematical Problems in Engineering 2015 (2015) 876862.
[21] U. Schlegel, D. V. Lam, D. A. Keim, D. Seebacher, Ts-mule: Local interpretable model-agnostic

explanations for time series forecast models, 2021. arXiv:2109.08438.
[22] Y. Wan, X. Gong, Y.-W. Si, Effect of segmentation on financial time series pattern matching, Applied Soft Computing 38 (2016) 346–359. URL: https://www.sciencedirect.com/science/article/pii/S1568494615006341. doi:https://doi.org/10.1016/j.asoc.2015.10.012.
[23] Y.-W. Si, J. Yin, Obst-based segmentation approach to financial time series, Engineering Applications of Artificial Intelligence 26 (2013) 2581–2596. URL: https://www.sciencedirect.com/science/article/pii/S0952197613001723. doi:https://doi.org/10.1016/j.engappai.2013.08.015.
[24] B. Håvardstun, C. Ferri, K. Flikka, J. A. Telle, XAI for time series classification: Evaluating the benefits of model inspection for end-users, in: Explainable Artificial Intelligence, 2024. URL: https://xaiworldconference.com/2024/, to be published.
[25] Z. J. Wang, J. W. Vaughan, R. Caruana, D. H. Chau, GAM coach: Towards interactive and user-centered algorithmic recourse, in: Proc. Conf. on Human Factors in Computing Systems„ ACM, 2023, pp. 835:1–835:20.
[26] J. Höllig, C. Kulbach, S. Thoma, Tsinterpret: A python package for the interpretability of time series classification, J. Open Source Softw. 8 (2023) 5220.
[27] W. Tang, L. Liu, G. Long, Interpretable time-series classification on few-shot samples, in: 2020 International Joint Conference on Neural Networks (IJCNN), IEEE, 2020, pp. 1–8.
[28] Y. Ming, P. Xu, H. Qu, L. Ren, Interpretable and steerable sequence learning via prototypes, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 903–913.
[29] 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, Hexagon-ML, The UCR time series classification archive, 2018.
[30] Z. Wang, W. Yan, T. Oates, Time series classification from scratch with deep neural networks: A strong baseline, CoRR abs/1611.06455 (2016). URL: http://arxiv.org/abs/1611.06455. arXiv:1611.06455.
[31] K. S. Gurumoorthy, P. Jawanpuria, B. Mishra, Spot: A framework for selection of prototypes using optimal transport, 2021. arXiv:2103.10159.
[32] I. Contributors, Interpretml: Fit interpretable models. explain blackbox machine learning., GitHub repository (2023). URL: https://github.com/interpretml/interpret.

7. Appendix

7.1. Pseudo code for DP algorithm

Define a 2-dimensional DP table $D[j, p]$ that stores the pth least costly simplification for the points $p_1, ..., p_j$ ending at p_j. Set $D[1, 1] = 0$.
Compute all $D[j, p]$ for $p \geq 1$ as follows:
for $j = 1$ to $n - 1$ **do**
    Initialize an empty heap H_j to store triples (value, index, rank) with value being the key.
    for $i = 1$ to $j - 1$ **do**
        H_j.add$(\alpha \cdot err(i, j) + \beta + D[i, 1], i, 1)$
        if $i \neq 1$ **then**
            H_j.add$(\alpha \cdot err(1, i, j) + \beta, i, -1)$
        end if
    end for
    for $p = 1$ to q **do**
        $(v_m, index_m, rank_m) = pop_{min}(H_j)$
        $D[j, p] = v_m$
        if $rank_m \neq -1$ **then**
            H_j.add$(\alpha \cdot err(index_m, j) + \beta + D[index_m, rank_m + 1], index_m, rank_m + 1)$
        end if
    end for
end for
To ensure that the last segment is not required to stop at p_n, but instead can be a continuation of two other points p_i and p_j, go over all pairs of points and evaluate the error if we continue them to the end.
Store these possibilities on a heap H_n and extract the q best, similar to what we did previously.
Initialize an empty heap H_n to store quadruples (value, index, rank, end) with value being the key.
for $j = 1$ to n **do**
    for $i = 1$ to $j - 1$ **do**
        H_n.add$(\alpha \cdot err(i, j, n) + \beta + D[i, 1], i, 1, j)$
        if $i \neq 1$ **then**
            H_n.add$(\alpha \cdot err(1, i, j, n) + \beta, i, -1, j)$
        end if
    end for
end for
for $p = 1$ to q **do**
    $(v_m, index_m, rank_m, end_m) = pop_{min}(H_n)$
    $D[n, p] = v_m$
    if $rank_m \neq -1$ **then**
        H_n.add$(\alpha \cdot err(index_m, end_m, n) + \beta + D[index_m, rank_m + 1], index_m, rank_m + 1, end_m)$
    end if
end for