Universidade Estadual de Londrina

VITOR DE CASTRO SILVA

EXPLAINABLE TIME SERIES TREE: AN EXPLAINABLE TOP-DOWN TIME SERIES SEGMENTATION FRAMEWORK

LONDRINA
2024

VITOR DE CASTRO SILVA

EXPLAINABLE TIME SERIES TREE: AN EXPLAINABLE TOP-DOWN TIME SERIES SEGMENTATION FRAMEWORK

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

Supervisor:	Prof. Dr. Bruno Bogaz Zarpelão
Co-supervisor:	Prof. Dr. Sylvio Barbon Júnior

LONDRINA
2024

Ficha de identificação da obra elaborada pelo autor, através do Programa de Geração Automática do Sistema de Bibliotecas da UEL

S586e Silva, Vitor de Castro .
Explainable Time Series Tree : An explainable top-down time series segmentation framework / Vitor de Castro Silva. - Londrina, 2024.
83 f. : il.

Orientador: Bruno Bogaz Zarpelão.
Coorientador: Sylvio Barbon Júnior.
Dissertação (Mestrado em Ciência da Computação) - Universidade Estadual de Londrina, Centro de Ciências Exatas, Programa de Pós-Graduação em Ciência da Computação, 2024.
Inclui bibliografia.

1. Séries temporais - Tese. 2. Mudança de Conceito - Tese. 3. Segmentação de Séries Temporais - Tese. 4. Regressão Simbólica - Tese. I. Zarpelão, Bruno Bogaz. II. Júnior, Sylvio Barbon. III. Universidade Estadual de Londrina. Centro de Ciências Exatas. Programa de Pós-Graduação em Ciência da Computação. IV. Título.

CDU 519

VITOR DE CASTRO SILVA

EXPLAINABLE TIME SERIES TREE: AN EXPLAINABLE TOP-DOWN TIME SERIES SEGMENTATION FRAMEWORK

Dissertação apresentada ao Programa de Mestrado em Ciência da Computação da Universidade Estadual de Londrina para obtenção do título de Mestre em Ciência da Computação.

BANCA EXAMINADORA

Orientador: Prof. Dr. Bruno Bogaz Zarpelão
Universidade Estadual de Londrina

Prof. Dr. Daniel dos Santos Kaster
Universidade Estadual de Londrina

Prof. Dr. Rodrigo Sanches Miani
Universidade Federal de Uberlândia

Londrina, 19 de Abril de 2024.

Dedico este trabalho à todos que estiveram comigo durante este período da minha vida, oferecendo apoio das mais diversas formas sempre que precisei.

ACKNOWLEDGEMENTS

Agradeço aos professores que me orientaram ao longo do trabalho, ajudando a organizar as minhas ideias e guiando a minha escrita.

Agradeço ao Instituto de Desenvolvimento Rural do Paraná por ter me fornecido o material necessário para a pesquisa e realização dos experimentos.

Agradeço à Fundação Araucária pela oportunidade de me dedicar exclusivamente a executar projetos relacionados à minha área de interesse ao longo do mestrado.

Por fim, agradeço ao Centro de Inteligência Artificial no Agro por ter me apresentado a diversos profissionais incríveis.

"O todo é maior que a simples soma de suas partes" (Aristóteles)

SILVA, V. C.. **Explainable Time Series Tree: An explainable top-down time series segmentation framework**. 2024. 83p. Master's Thesis (Master in Science in Computer Science) – State University of Londrina, Londrina, 2024.

ABSTRACT

A wide range of Machine Learning algorithms can model time series to address classification, forecasting, and clustering problems. However, time series may exhibit characteristics that complicate these tasks, such as repeating patterns and seasonal variations. Time series segmentation could be a solution to these problems, but current approaches need to be improved. Most of them employ linear regression to solve problems such as detecting changes in a series' behaviour, bypassing tools specifically designed for these challenges, such as change detectors. Moreover, explainability is seldom taken into account during time series segmentation. To automatically identify different time series patterns using appropriate techniques while leveraging explainability, we proposed the eXplainable Time Series Tree (XTSTree). XTSTree divides a time series into a binary tree, hierarchically splitting it according to a criterion based on change detectors, ideally finding a cutting point that creates the two most different sub-series. The segmentation process continues until it reaches a stopping criterion, which relies on a stationarity test that assesses whether the series has a sufficiently homogeneous behaviour. Based on well-behaved segments, XTSTree paves the way for a more comprehensive pattern explanation and also supports the application of explainable approaches. We applied XTSTree on several real-life time series to isolate the series' different behaviours. To evaluate the effectiveness of our method, we used an implementation of Symbolic Regression using genetic programming to find the best representation of the time series and its splits using algebraic expressions, comparing the differences before and after XTSTree. We show an improvement in terms of formula complexity, improving the model accuracy compared to the original time series.

Keywords: Concept Drift, Time Series, Meta-learning, Time Series Segmentation, Symbolic Regression

SILVA, V. C.. **Explainable Time Series Tree: Um framework de segmentação *top-down* de séries temporais explicável**. 2024. 83f. Dissertação (Mestrado em Ciência da Computação) – Universidade Estadual de Londrina, Londrina, 2024.

RESUMO

Existem diversos algoritmos de Aprendizado de Máquina para modelar séries temporais para problemas de classificação, previsão e clusterização. Entretanto, séries temporais podem apresentar características que complicam essas tarefas, como padrões recorrentes e variações sazonais. A segmentação de séries temporais pode ser uma solução para esses problemas, mas as abordagens existentes precisam ser melhoradas. Várias delas utilizam regressão linear para solucionar problemas como detecção de mudanças no comportamento da série, ignorando ferramentas desenvolvidas especificamente para resolver esse tipo de problema, como detectores de mudança. Além disso, o conceito de explicabilidade raramente é abordado na segmentação de séries temporais. Para identificar diferentes padrões de séries temporais usando técnicas apropriadas e ao mesmo tempo prover explicabilidade, nós criamos a *eXplainable Time Series Tree*, ou XTSTree. A XTSTree transforma uma série temporal em uma árvore binária, dividindo-a de forma hierárquica de acordo com um critério baseado em detectores de mudança e encontrando um ponto de corte que cria as duas sub-séries mais diferentes entre si. O processo de segmentação continua até que chegue em um critério de parada baseado em um teste de estacionariedade que indica que a série tem um comportamento suficientemente homogêneo. A XTSTree abre caminho para uma explicação mais compreensível de padrões e também oferece suporte para o uso de outras abordagens explicáveis. Nosso estudo aplicou a XTSTree em diversas séries temporais reais para isolar os seus diferentes comportamentos. Para avaliar a eficácia da XTSTree, nós usamos uma implementação de Regressão Simbólica usando programação genética para encontrar a melhor representação da série e suas divisões usando fórmulas algébricas, e comparamos as diferenças em diversas métricas antes e depois da XTSTree. Mostramos uma melhora em complexidade da fórmula, melhorando a acurácia do modelo quando comparado com a série temporal original.

Palavras-chave: Mudança de Conceito, Séries Temporais, Meta-learning, Segmentação de Séries Temporais, Regressão Simbólica

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

XTSTree	eXplainable Time Series Tree
SR	Symbolic Regression
GP	Genetic Programming
PH	Page-Hinkley
ADF	Ad-Fuller
ML	Machine Learning
AI	Artificial Inteligence
PLR	Piecewise Linear Representation
IDR-PR	Instituto de Desenvolvimento Rural do Paraná
RMSE	Root Mean Square Error
GDPR	European General Data Protection Regulation
LGPD	Lei Geral de Proteção de Dados

CONTENTS

1 INTRODUCTION

Temporal data is a common type of data that has been increasingly used in several tasks from different areas, such as e-health [2], Internet of Things systems [3], and autonomous vehicles [4]. The most common way to represent temporal data for modelling and prediction is through time series. Several studies have already demonstrated that time series classification and prediction are suitable to building relevant applications. Doki et al. [5] created a model for human behaviour which was subsequently used to indicate and suggest actions for other humans. In another study, Chiang & Dey [3] use data from a blood pressure monitor to predict blood pressure levels and point out which routine changes caused the most amount of change in blood pressure.

One problem these techniques face is the change in data behaviour through time [6], which often happens due to the nature of data, such as a temperature drop during a rain period or a heart rate rise during stress. Often defined as *concept drift*, these changes require retraining models along the time series, or assuring models are suited for every behaviour present. Moreover, these behaviours can repeat over time, such as in seasons over a year or Internet traffic over a week. Identifying areas with different behaviours can ease time series modelling since different behaviours can be treated individually. One way to identify similar behaviours in a time series is through time series segmentation [7].

Time series segmentation is the process of obtaining segments of a time series that either have similar behaviour or behave differently from the rest of the series [7]. There are three main types of segmentation algorithms: sliding window algorithms, which create and repeatedly increase the size of a window from the start of the series until it reaches an error threshold, repeating the process with a new window at that point [8]; bottom-up algorithms, which partition the time series into several small pieces and merges the least influential ones until a threshold is met [9]; and top-down algorithms, which finds the best cutting point in a series, and then repeats the process for each sub-series that falls above a user-defined threshold [10].

Currently, proposed top-down algorithms often derive the error value and stopping criterion from a linear regression model [10]. Furthermore, the best cutting point is either tied to the linear regression or is found through an exhaustive search of possible partitions [11]. In the former, the position where the series deviates the most from the expected value given by the linear regression model is chosen as a split point. This process is similar to detecting a concept drift, a problem for which there are existing tools, such as change detectors [12].

In this paper, we present the eXplainable Time Series Tree, or XTSTree, a top-

down segmentation algorithm that uses a statistical test as a stopping criterion for the segmentation process, and a change detector to find the cutting position. Unlike other top-down segmentation methods, our approach uses a change detector to identify the best cutting positions, instead of relying on an error function based on linear regression. The primary goal of the XTSTree framework is to provide a tree-based structure that enables in-depth comprehension of the diverse patterns within a time series in an explainable manner. By representing each leaf as a contiguous pattern, XTSTree facilitates intuitive modelling and analysis of individual segments. This tree-based approach ensures interpretability through the hierarchy of cuts, allowing users to understand and interpret the diverse range of patterns captured within the time series. Additionally, XTSTree is scalable due to the binary tree's low memory cost and the change detector's efficiency, making it suitable for processing large and complex datasets. Since it treats the sub-series as an individual time series, it allows for the integration of advanced techniques and algorithms, further enhancing its accuracy and versatility in real-world time series analysis scenarios. In this study, we created an implementation using the Page-Hinkley change detector (PH) [12] and the Augmented Dickey-Fuller test (ADF) [13].

To test the effectiveness of XTSTree, we segmented several different weather-related time series using the original XTSTree, two other segmentation methods and one stopping criteria as baselines, and two implementations of the top-down segmentation algorithm. Afterwards, we implemented Symbolic Regression (SR) using Genetic Programming (GP) to create algebraic formulas that describe the series' behaviour as a whole and segmented. Finally, we compared formula complexity, prediction errors, number of cuts made, execution time and an entropy metric. All series were provided by the Institute of Rural Development of Paraná (IDR-PR). This work is part of the Artificial Intelligence Center for Agriculture (CIA-Agro). The main goal of the CIA-Agro is to be a transforming agent for social and technical development to provide wealth and well-being through Artificial Intelligence (AI) in agriculture.

The manuscript is organized as follows. Chapter 2 presents definitions of concepts related to explainability, time series segmentation, and tools that are part of the methodology. Chapter 3 discusses related work in the field of time series segmentation and presents a comparison between the proposed solution and works from the state-of-the-art. Chapter 4 introduces the proposed approach and contains a toy experiment detailing the step-by-step process through the segmentation of a synthetic time series, showing formulas obtained from SR before and after segmentation. Chapter 5 presents experiments comparing XTSTree with two variations of top-down segmentation algorithms and other developed baselines, as well as a case study applying XTSTree in a real time series. Finally, Chapter 6 makes the concluding remarks.

2 BACKGROUND

In this section, we explain concepts relevant to understand the segmentation method, the stopping criterion implemented by XTSTree, and the evaluation method for its effectiveness.

2.1 Time Series

Time series is a commonly used type of data that has its data points ordered over time. This kind of data can be divided into univariate time series and multivariate time series. A univariate time series is a sequence of single values along time, such that $T_u = (t_0, t_1, t_2, ..., t_n)$. These values can be of any type but are most commonly numerical, such as air temperature or monthly number of sales. Multivariate time series, on the other hand, are sequences of multiple values representing different information at the same timestamp, such that $T_m = ((t_0, u_0, v_0, w_0), (t_1, u_1, v_1, w_1), ..., (t_n, u_n, v_n, w_n))$. This type of time series enables us to monitor multiple pieces of information simultaneously at the expense of simplicity.

Time series can provide a plethora of useful information, including trends, seasonality, and stationarity. The trend of a series is defined as a continuous increase or decrease in the series' values, while seasonality is a periodic repetition of a pattern. A time series is considered stationary if its mean and variance are constant, meaning that neither trend nor seasonality are present. Stationarity is an important characteristic of time series because dealing with non-stationary series requires understanding multiple variables related to their trend and seasonality, while a stationary series will have predictable behaviour independently of the timestamp observed. For this reason, most tools applied in time series analysis assume the series is stationary, requiring that some type of treatment is carried out otherwise.

A series' stationarity can be determined by visually observing its behaviour or through several different specialized tools, such as stationarity tests and unit root tests.

2.1.1 Stationarity tests and unit root tests

Stationarity tests are a type of statistical test used to determine if a series is stationary. Its null hypothesis is that stationarity is present, the alternative indicating its absence [14]. A complementary test to stationarity tests are unit root tests, where the null hypothesis is the presence of a unit root, indicating that the series is not stationary.

A unit root is a characteristic of a time series that indicates the value of an element y_t of a time series is dependent on t. One way to represent a time series is through a

combination of its previous values. A model that determines the value at a timestamp t using previous values is called an autoregressive model. These models are written as $y_t = \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + e_t$, where p is the order of the autoregressive model and e_t is a random number with mean 0 and variance σ^2, i.e. $N(0, \sigma^2)$ [15].

Let us assume a time series created by the autoregressive model $y_t = \phi y_{t-1} + e_t, t = 1, 2, ...$, $\phi \in \mathbb{R}$ and e_t being a random number with mean 0 and variance σ^2. If $|\phi| < 1$, since e_t has a fixed variance, there is a value of y_{t-1} where $|\phi y_{t-1} + e_t|$ is unlikely to be greater than $|y_{t-1}|$. Therefore, Y will converge to a stationary time series of fixed mean as $t \to \infty$ [13], and is said not to have a unit root.

Similarly, if $|\phi| > 1$, there is a value of y_{t-1} where $|\phi y_{t-1}| - |e_t|$ is unlikely to be smaller than $|y_{t-1}|$, meaning the values of Y will change at an exponential rate. Finally, if $|\phi| = 1$, Y will not converge towards a fixed mean, and its value at y_t will be $t * N(0, \sigma^2)$, meaning the value of y_t depends on the timestamp t, the series does not tend to converge to a fixed mean, and the series has a unit root. This behaviour is known as a random walk, and in such cases it may be appropriate to differentiate the time series.

One popular test used to determine if a time series has a unit root is the Augmented Dickey-Fuller test [16]. It is possible to estimate the value of p to determine if Y is stationary through the formula $\hat{p} = \frac{\sum_{t=1}^{n} y_t y_{t-1}}{\sum_{t=1}^{n} y_{t-1}^2}$. By applying a statistical test using $\hat{p}$ and comparing the result to the corresponding critical value as calculated by Mackinnon [17], it is possible to ensure the correctness of the estimation, and consequently the stationarity of the series. The ADF test has been thoroughly studied and improved upon, and there are several open-source implementations.

Figure 1 shows a comparison between a stationary and a non-stationary series, with their respective values for the ADF test. The top graph is a stationary series created using the autoregressive model $y_t = 0.8 y_{t-1} + N(0, 1)$, while the bottom graph is a non-stationary series created with the model $y_t = 0.8 y_{t-1} + N(0, 1)$. It is evident that the top graph has a constant variance and mean, while the bottom graph's mean varies as the timestep increases, and the results for the ADF test confirm the presence or absence of a unit root.

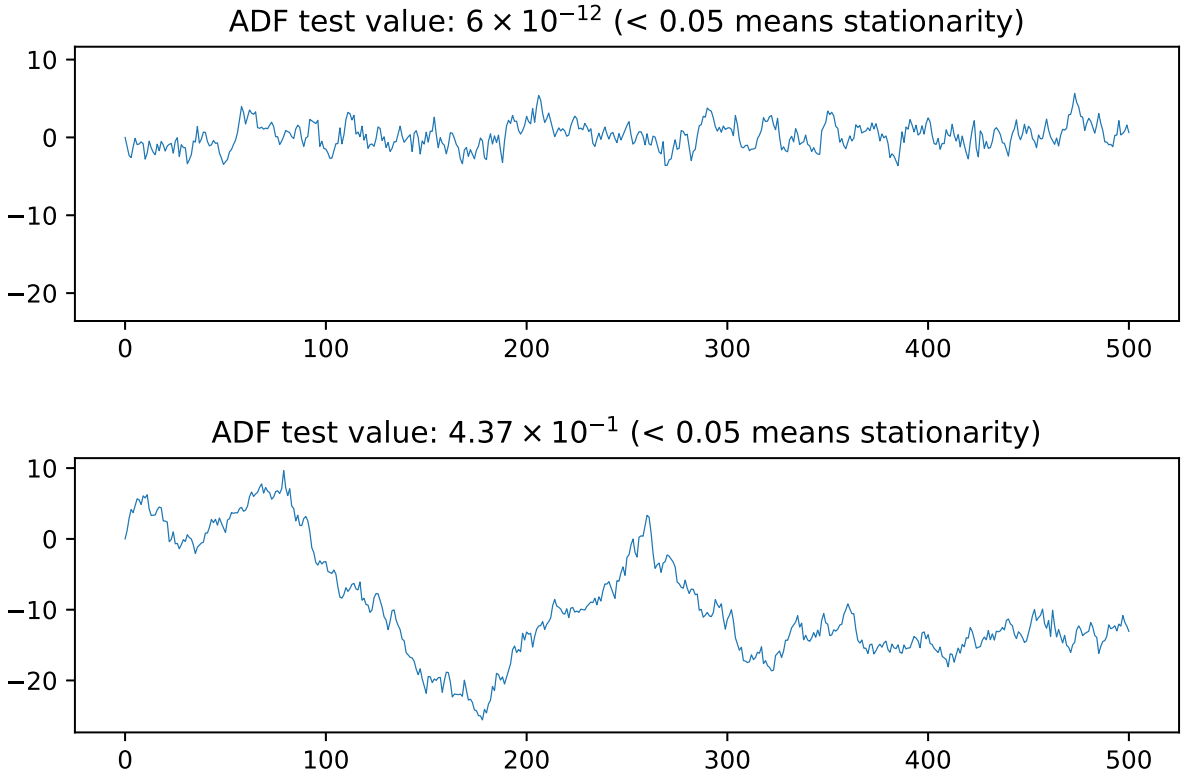


Figure 1 – Comparison between stationary and non-stationary series. Both series were generated using the autoregressive model $y_t = py_{t-1}+e, y_0 = 0$ and $e = N(0, 1)$. Top and bottom figures use $p = 0.8$ and $p = 1$ respectively.

2.1.2 Change detectors

Time series are mostly used to monitor variables over time, leveraging the variations in values to predict and classify future events, make decisions relating to management, and various other tasks. However, these variations can add up as time passes, resulting in a different mean, trend, or behaviour and reducing the accuracy of these tasks. Therefore, models trained using a past section of a time series can become outdated, resulting in an event called *concept drift* [18]. Figure 2 shows an example of a series, highlighting a period of concept drift where it alternates between two different behaviours.

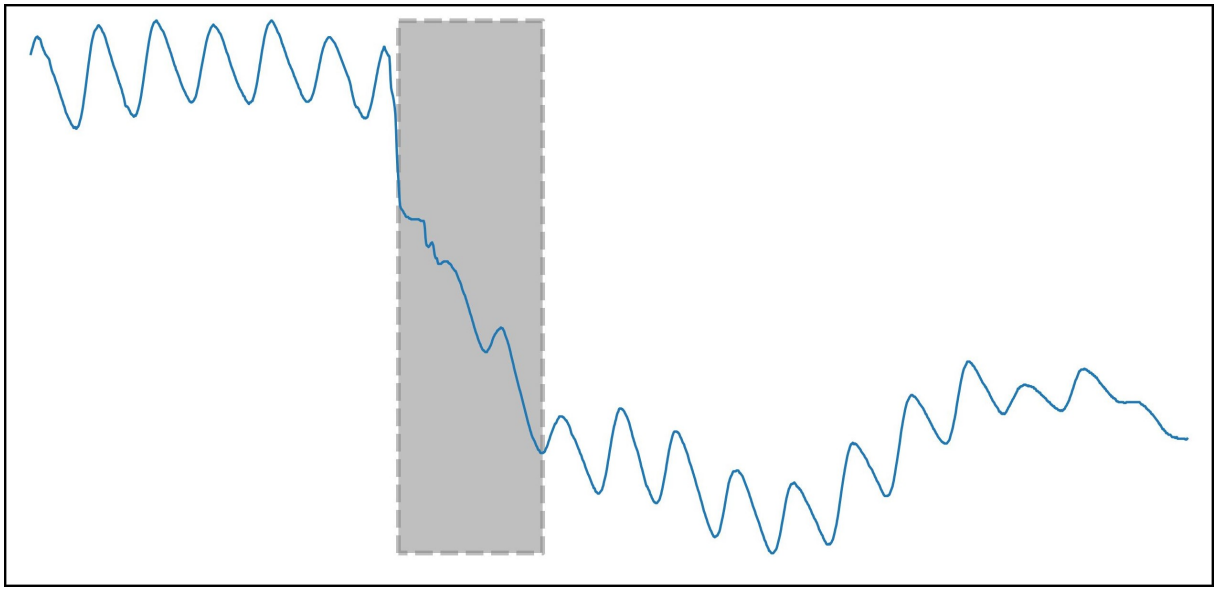


Figure 2 – Time series with a concept drift. Highlighted in grey is a period where the series changes behaviours before it stabilizes again.

Several studies have been made to identify and work around these changes and ensure that a model for the time series stays updated [19, 18], with one of the most common algorithms utilized being the change detector. These algorithms monitor a time series, possibly in real-time, and when a threshold for changes in the time series is met, it triggers an alarm indicating a sufficient amount of change has occurred in the expected behaviour of the series to consider a concept drift. This alarm can be used as an automated trigger to start a retraining process for models monitoring the time series.

The Page-Hinkley test is an example of a change detector for univariate time series. It continuously observes values from a given time series while monitoring the accumulated difference between the current value and the mean of the preceding ones. This difference is represented by the equation $m_T = \sum_{t=1}^{T}(x_t - \overline{x}_T - \delta)$, where t is the observed timestamp, $\overline{x}_T$ is the mean of the previous values, and δ is an expected value of variability [20]. With each new value observed, the value of m_T is reduced by a factor of $\alpha \in (0, 1]$, acting as a forgetting factor to reduce the impact of older observations.

When the value of m_T exceeds a predefined threshold value, a change is detected, m_T is set to 0 and the monitoring process restarts from that timestamp. Some implementations only monitor positive changes, while others monitor both positive and negative. A degree of explainability of the changes detected can be achieved due to the fairly concise behaviour of the PH detector since the series and difference between mean and threshold can be monitored. Therefore, abrupt differences may indicate a drastic change in behaviour or an outlier, while more subtle changes over time may indicate a smoother and longer transition.

2.1.3 Time Series Segmentation

The main goal behind data analysis is to extract as much relevant information as possible from data, identifying patterns and other significant features. This process applied to large amounts of data is called *data mining*. Time series segmentation is a form of data pre-processing for time series *data mining*. Its main goal is to divide a time series made of various observations into smaller segments where observations belonging to the same segment are similar in some manner. This segmentation can be beneficial in many ways, such as facilitating the classification of different periods of a series or identifying different behaviours to deal with possible concept drifts [7].

One example where it is desired to segment a time series is shown in Figure 2. If done correctly, the segmentation could create separate subseries, which then could be treated differently. Figure 3 shows one possible segmentation of Figure 2, where three subseries with well-defined behaviour are defined, isolating the concept drift stretch, and these segments could subsequently be addressed individually.

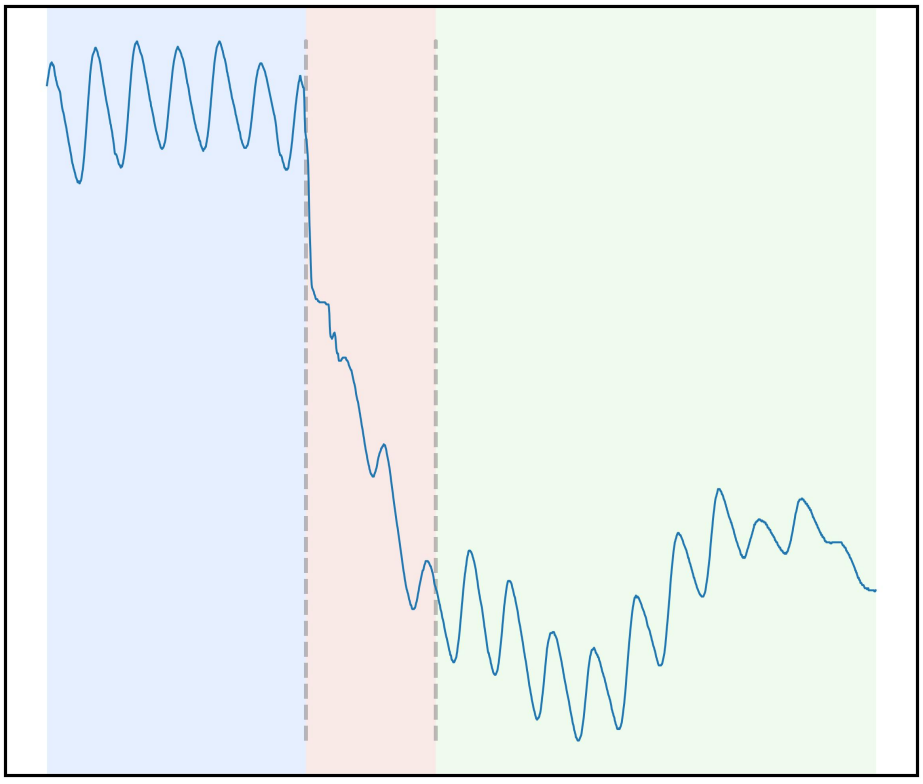


Figure 3 – Example of a valid segmentation of a series with concept drift. Each colour represents a different subseries, and dashed lines are the divisions between subseries.

Given a time series X such that $X = [x_1, x_2...x_n], x_i \in R$, the segmentation process aims to find $k \in N$ distinct points such that $k_0 = 0, k_K = N$ and $F(X) \geq F(X_{[k_i,k_{i+1}]}), i \in [1...K]$, where F is a modelling technique that outputs a score or error value. There are

several modelling techniques used in time series segmentation, such as black-box classification models or linear regression models. One implementation thoroughly explored is based on Piecewise Linear Representation (PLR). PLR is the process of reducing a time series to a sequence of lines [7], using mostly linear regression as a modelling technique. There are three main types of segmentation algorithms: sliding window; bottom-up; and top-down [11].

A bottom-up algorithm starts by dividing a series into several small segments. Then, the error value of each segment is calculated using linear regression and, for each adjacent segment, the error value of their combination is also calculated. The couple of segments that provide the lowest increase in error when compared to their values are merged, and the process repeats itself until a threshold value for the series' error is exceeded or a minimum number of segments remain. Algorithm 1 describes a bottom up algorithm. Figure 4 shows a step-by-step of the segmentation process of a bottom-up algorithm. It starts by creating several small segments, then joining pairs until a desired segmentation is achieved.

Algorithm 1: Bottom-Up Pseudocode

```
1  Function bottom_up(series) begin
       Data: series: Series to be segmented
       Input: threshold: Threshold set by user
       Input: min_segments: Minimum number of segments required
2      segments ← []
3      create segments of size two for all elements of the series S
4      calculate the error for all segments E_s
5      calculate the error for all couple of adjacent segments when combined E_c
6      error_series ← ∑ E_s
7      while error_series ≥ threshold and length(segments) > min_segments do
8          find segments i, j and combine segment ij such that min(E_s[i] + E_s[j] − E_c[ij])
9          remove segments i, j from S
10         add segment ij to S
11         error_series ← ∑ E_s
12     end
13     return segments
14 end
```

Sliding window algorithms work by setting an anchor at the start of the time series. That anchor is also the beginning of the first segment. Then, elements from the time series are added to the segment while fitting a linear regression model to it. This happens until the linear regression error exceeds a predefined threshold, at which point the end of the segment is set. Afterwards, the anchor is set to the timestamp after the last added element, a new segment is initiated and the process repeats itself [11]. Sliding window algorithms belong to the class of online algorithms, meaning that the time series does not need to be finite and new elements can be added to it. Algorithm 2 represents a sliding window algorithm. Figure 5 represents a step-by-step of the sliding window algorithm. The grey area represents the sliding window, and each differently coloured stretch is a different segment.

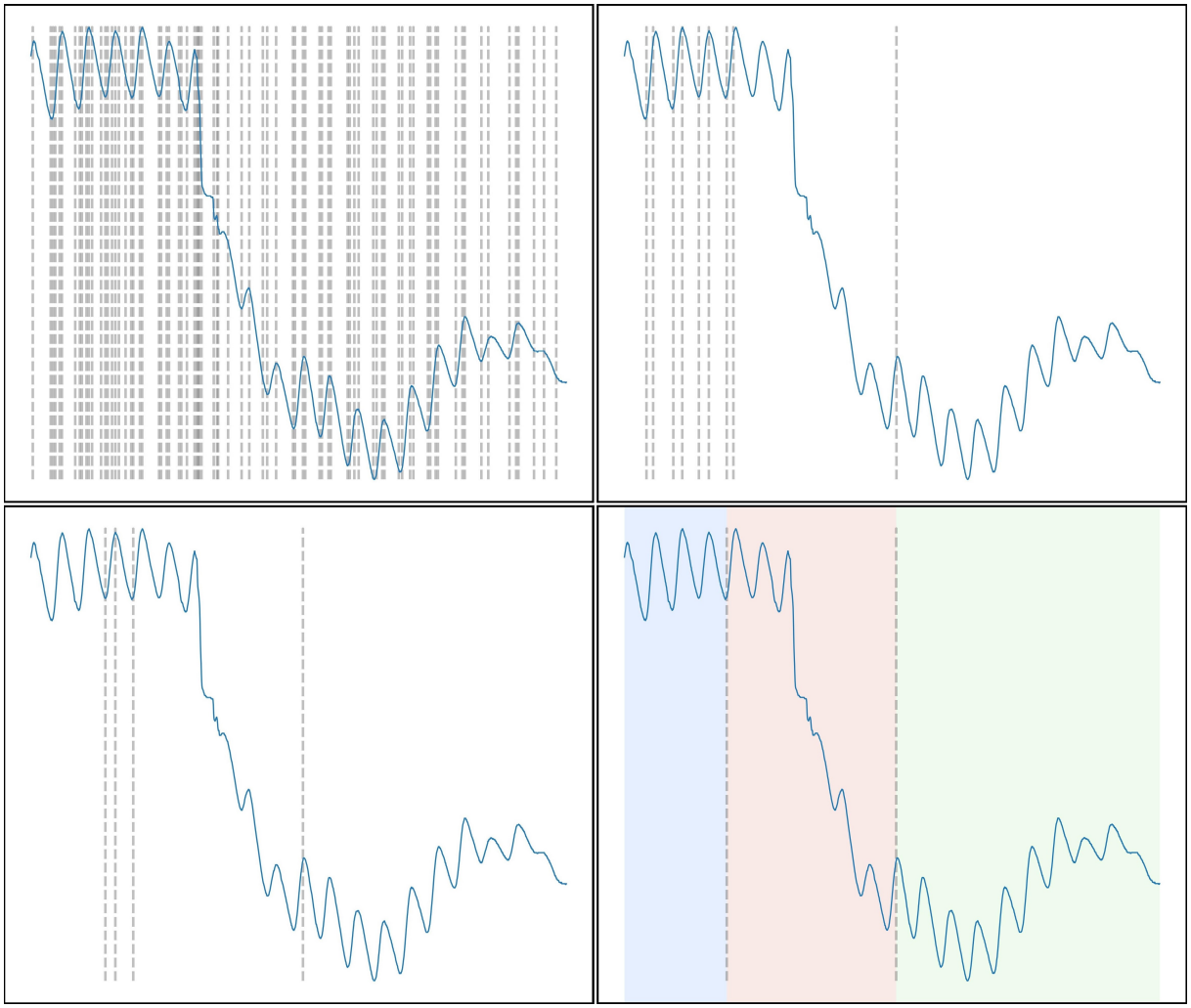


Figure 4 – Example of the segmentation process of a bottom-up algorithm. From left to right, the figures have 80, 10, 5 and 3 segments respectively. The bottom-up algorithm creates several small segments and aims to reduce the number of segments repeatedly until a satisfactory result is achieved.

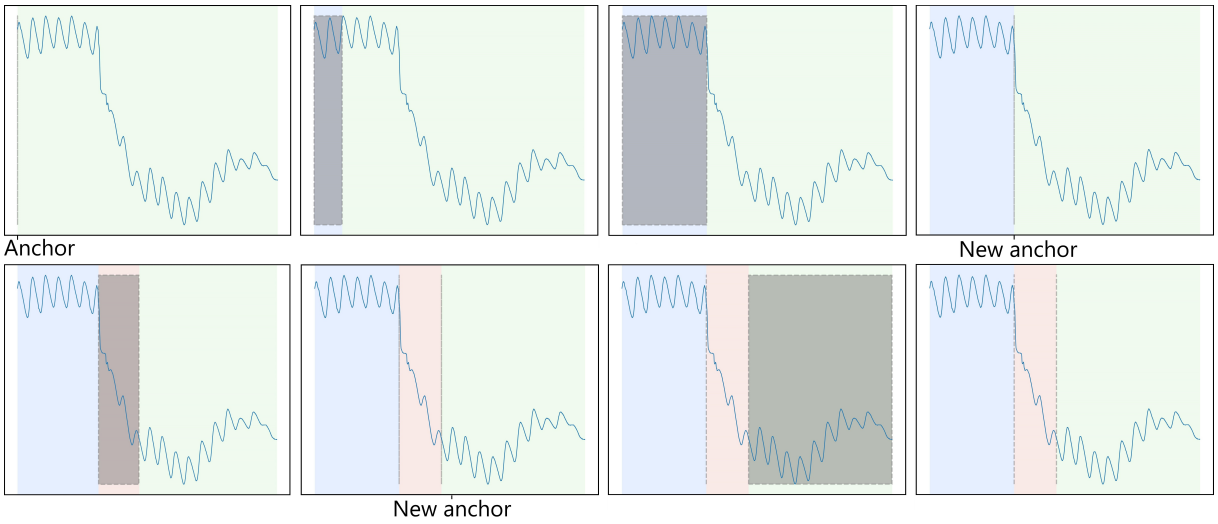


Figure 5 – Example of the segmentation process of a sliding window algorithm. The sliding window starts at the beginning of the series, expanding along it. When a certain threshold is met, it defines the segment and starts a new window.

Algorithm 2: Sliding Window Pseudocode

```
1  Function sliding_window(series) begin
       Data: series: Series to be segmented
       Input: threshold: Threshold set by the user
2      start ← 0
3      end ← 0
4      error ← 0
5      segments ← []
6      for element in series do
7          while error ≤ threshold do
8              end ← end + 1
9              calculate error for series[start:end]
10         end
11         add series[start:end] to segments
12         start ← end
13         error ← 0
14     end
15     return segments
16 end
```

A variation of sliding window algorithms referred to as Sliding Window And Bottom-up was proposed by Keogh et al. [11]. It consists of creating a buffer window large enough to fit a small number of segments, usually 5 to 6, at the beginning of the series. The bottom-up algorithm is applied to the buffer and the segment at the start of the buffer is removed as a segment. The same number of elements removed are added to the buffer, sliding the buffer window, and the bottom-up algorithm is reapplied. This process repeats until the complete series is segmented.

Top-down algorithms employ an opposite approach to the bottom-up. They first take the complete series as a segment and apply linear regression to it. If that segment produces an error value greater than the user-specified threshold, the segment is split in two at the timestamp that produces the largest amount of error. This process is repeated until every segment produces an error smaller than the user threshold [11]. Algorithm 3 describes a top-down algorithm. Figure 6 presents the step-by-step of a top-down segmentation. The series is defined as one segment, which is then divided into two subseries. This process repeats itself where the series does not have a defined behaviour.

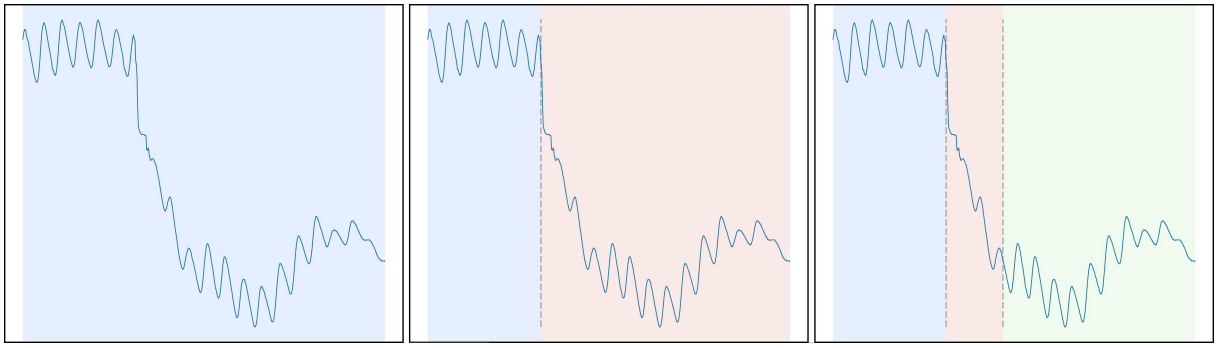


Figure 6 – Example of the segmentation process of a top-down algorithm. Each different coloured background represents a segment, and the dashed lines are the divisions between segments.

Algorithm 3: Top-Down Pseudocode

```
1  Function top_down(series) begin
       Data: series: Series to be segmented
       Input: threshold: Threshold set by the user
2      segments ← []
3      calculate error E for series
4      if E > threshold then
5          find position with greatest error i in series
6          add top_down(series[0 : i]) to segments
7          add top_down(series[i : length(series)]) to segments
8          return segments
9      else
10         return series
11     end
12 end
```

While there are other variations of segmentation algorithms [21, 22], most of them employ a strategy similar to the bottom-up, top-down or sliding window algorithms, with their greatest difference being a modelling technique other than linear regression.

2.2 Explainability and Interpretability

Machine learning (ML) has been increasingly applied to several problems due to its ability to outperform humans in both speed and accuracy. However, there are certain tasks where a decision made by a model can be severely impactful. If a bank uses an ML model for background checks and denies someone a bank loan, that person may demand an explanation from the bank. A medical diagnosis becomes more valuable if it is accompanied by a justification, which may even assist in treatment. Situations like these have become even more relevant since the European General Data Protection Regulation (GDPR), the Brazilian *Lei Geral de Proteção de Dados* (LGPD) and other similar regulations have taken effect. These regulations legally require that companies justify decisions made by models if requested.

The development of explainability has been primarily driven by situations like

these. Models that are easier to interpret have become more valuable, and there are various studies that aim to explain black-box models, either through input examples that clarify a model's behaviour, methods that grade data features in order of importance or surrogate models that try to emulate a black-box model while being easily explainable.

The concept of *Explainability* and *Interpretability* is still not well defined. Kim et al. [23] state that a method is interpretable if a user can correctly and efficiently predict the method's results. Miller [24] understands interpretability as "the degree to which an observer can understand the cause of a decision". Miller also defines explainability as a combination of three processes: the cognitive process, where the causes for an event are identified; the product, which is a subset of relevant causes from the previous step; and the social process, which dictates how the knowledge obtained from the product is transferred to a person. According to the author, since both definitions achieve the same final objective, they are used interchangeably.

Nauta et al. [25] define the concept of an explanation in the context of explainable AI as "... a presentation of (aspects of) the reasoning, functioning and/or behaviour of a machine learning model in human-understandable terms". In the context of this definition, reasoning is the process behind a decision, functioning is how data is stored and manipulated inside a model, and behaviour is how the model works from an outside view, such as what input created a given output. Molnar [26] also uses explainability and interpretability indistinguishably, defining the interpretability of a model as its capacity to make its decisions easy to understand. In this work, both terms are interchangeable.

2.2.1 Interpretable Models

According to Molnar [26], there are two main ways to employ explainability in a ML model: through the structure of the model or by using some method after training. Some models are explainable by default due to their simple structure, such as linear, logistic and symbolic regression and decision trees.

Decision trees are a type of tree-based model that tries to find several threshold values to classify data points. It chooses the values, features and number of decisions or nodes it should use. The main advantage of tree-based models in terms of explainability is the ability to interpret the model by visually looking at the nodes and used values [27]. Figure 7 shows an example of a decision tree that differentiates observations between the classes human, bird, cat and crocodile. Each non-leaf node represents a "question" that divides observations between classes, and the leaf nodes are the possible classes. In Figure 7, it first checks the number of legs of an observation, and based on that it either sees if it has feathers or if it likes water, assigning a class to it afterwards.

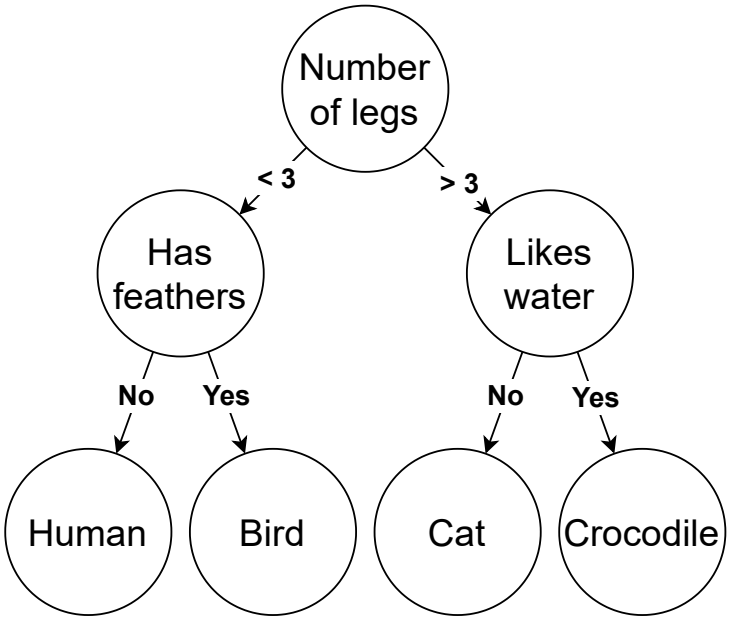


Figure 7 – Example of a decision tree. Each node represents a question, and each connection is a possible answer.

Since decision trees have well-defined thresholds at each node, it is easy to predict the output of the model given the input. In Figure 7, if an observation has two legs and no feathers, the model's output will be human. It is also easy to change the values of selected features and understand if and why the final output changed. For example, by adding feathers to an observation classified as human, it is apparent that the output will change to the bird class, therefore we can say that the difference between a human and a bird is the presence of feathers.

By looking at the visual structure of a tree, one can also understand certain relations between features, and it is possible to calculate the importance of each feature, making it easier to understand what features have more influence on the output [28]. Suppose that the model described in Figure 7 had more bird-like classes and features such as feather colour, wing size, beak length, etc. One could infer that, since these classes require more divisions, they are more complex. Also, if a node divides data evenly between child nodes, it means that the feature is important for the classification of that dataset. All of these characteristics make decision trees good explainable models.

Another type of easily interpretable models are formula-based models. Since these models use data features as input for a formula, it is simple to understand how each feature influences the model's output and to predict the output for a given input, two characteristics that make these models very explainable. One type of formula-based model are symbolic regression models.

Symbolic regression can be defined as a specialisation of linear regression. The goal of linear regression is to create a model that correlates a set of input variables and output

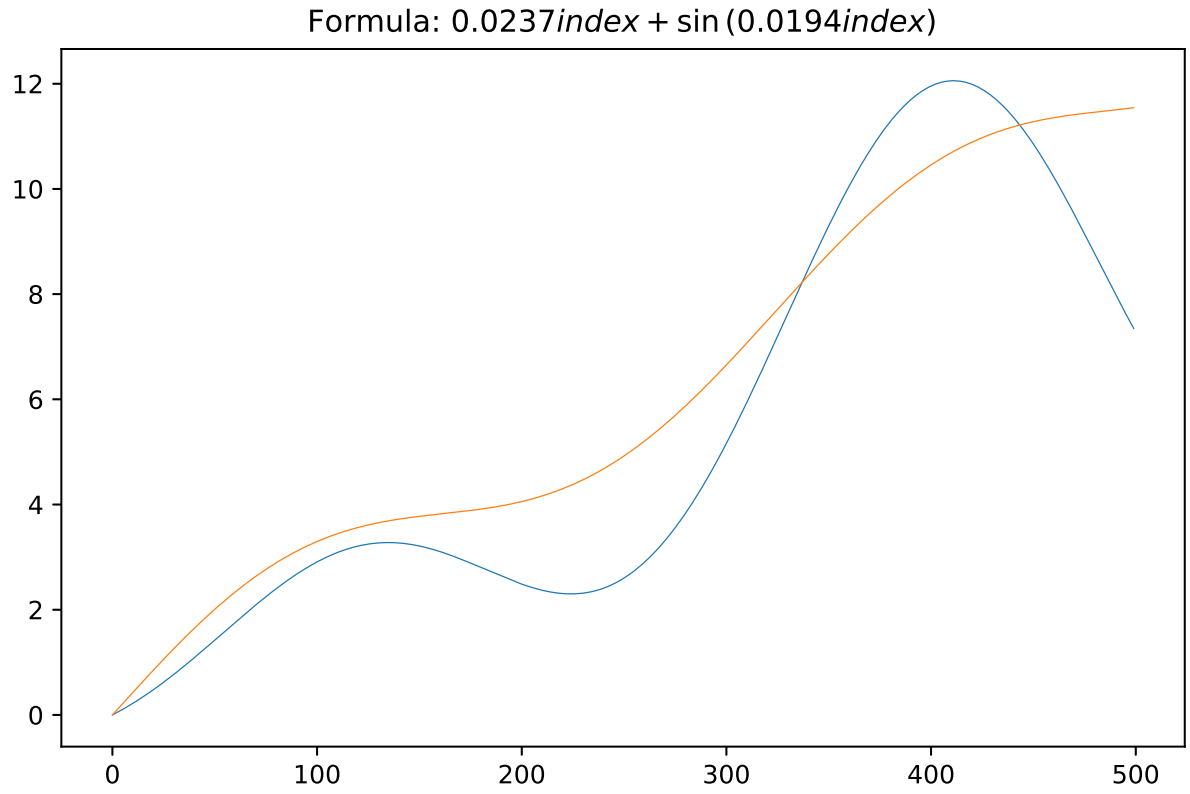


Figure 8 – Example of synthetic series and a representation using symbolic regression. The blue line represents the original series, while the orange line is an approximation using the formula shown in the image.

values using a linear formula, meaning that applying a linear regression will result in a linear formula that uses a set of variables as input and produces an output close to the set of output values. As linear regression can only create linear approximations, this type of regression can be ineffective in some situations. To address this issue, it is possible to let the algorithm find both the coefficients and operators of the formula. This process is known as *Symbolic Regression* [29]. Figure 8 shows an example of a synthetic time series and a possible representation using symbolic regression. The series was created using the formula $y_t = (t/100) * \sin t/50 + t/50$, and the approximation is shown in the image. It is worth noting that due to the nature of machine learning and genetic programming, the resulting formula is different, but it still emulates a similar behaviour to the originally used.

One common method to find the best formula for a set of variables in symbolic regression is by using Genetic Programming to create a syntax tree that represents the formula [30]. The GP algorithm creates a set of syntax trees where each tree node is either an operator or a terminal value. The input values are fed to the trees created and an error value is calculated based on the target variable. The best individuals are chosen to create the following generation, exchanging branches between each other, and repeating the first step. Figure 9 shows an example of a syntax tree. GP is both effective in terms

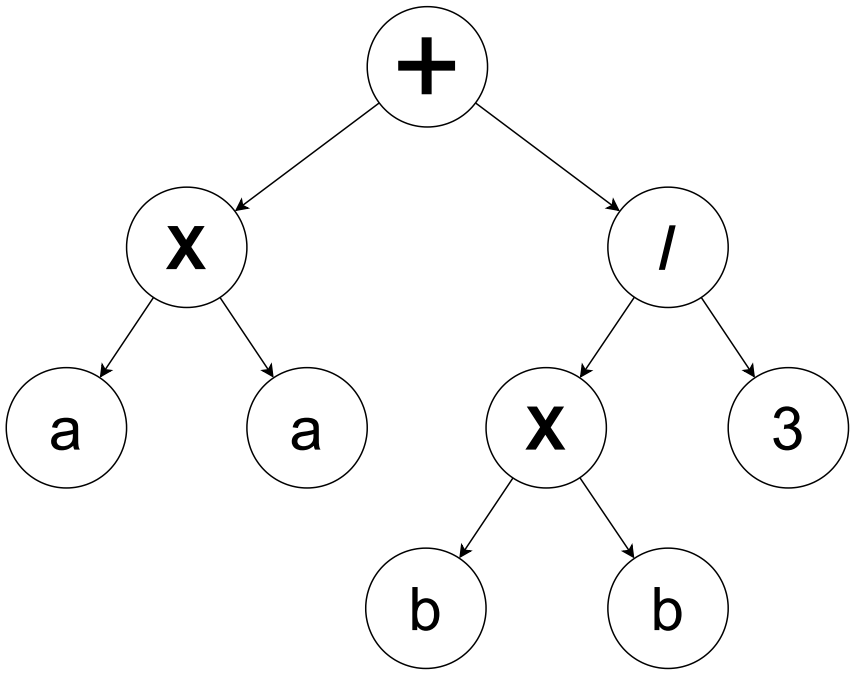


Figure 9 – Example of a syntax tree representing the formula $a^2 + b^2/3$. Leaf nodes represent constants or variables, while non-leaf nodes represent operations between its children.

of accuracy [31] and interpretability due to being able to produce simpler formulas that are easier for humans to understand when properly adjusted [32].

2.2.2 Explainable Methods

Other ML models may need extra steps to become explainable. One example is neural networks, which by default are not explainable, but have features and weights that can be analysed to explain it.

To explain ML models that are not interpretable, we can resort to Model-agnostic, Model-specific or Example-based methods. Model-agnostic are methods that work independently from the ML model. They can be divided into Local, meaning they explain individual outputs given by the ML model, and Global, meaning they explain the overall behaviour of the ML model concerning data [26, 27].

Global model-agnostic methods, in general, try to extract information from the features used to train the model, either ranking them, trying to measure how they affect the final result or trying to measure the relationship between different features [22]. Another type of global method is the global surrogate model, which consists of using an intrinsically explainable model to imitate the results of the black-box model [26]. Some examples of global methods are Partial Dependence Plot, Feature Interaction, Functional Decomposition and the Global Surrogate.

Local model-agnostic methods are diverse in terms of how they work. The indi-

vidual conditional expectation method varies the value of one of the features fed to the model while keeping the others unchanged to understand how that feature influences the final output [33]. Counterfactual explanations are another type of local model-agnostic methods which try to find the least amount of change it needs to make to an input to cause the most amount of change in the value of the output [26, 34, 35].

If data can be represented in a way humans can understand, explainability can be achieved through example-based methods. Similarly to model-agnostic methods, they can provide explainability to any machine learning model. Example-based methods consist of using one or more instances of data to explain the behaviour of the model [36]. One example of an example-based method is the k-nearest neighbours, where a prediction of value or class for an instance is the average of the k-nearest instances to it, and a possible explanation for the result lies in the k-nearest instances used [26].

As the name suggests, model-specific methods trade flexibility for a more specific interpretation of the model. Any interpretation derived from internal model attributes [27], such as linear regression coefficients, rules from a rule-based model [31] or split points for decision trees [28, 27], is model-specific. Explainable methods that work for only one type of model are also model-specific. To achieve explainability through model-specific methods in the context of neural networks, one can either understand what features or concepts the model learned, create an adversarial model to deceive the original model, or in the case of image data, find what pixels of an image are the most influential for the given result [26, 27].

2.2.3 Evaluating explainability

Explainability can be evaluated in three different layers: at the application, human and function levels [37]. Application-level evaluation involves employing domain experts to grade the output of a model. In this context, explainability is tied to how well the model output and explanations assisted a domain expert in a certain task. In a classification task, an explainable model will provide useful information as well as a label that the domain expert can use. Although this type of evaluation provides strong evidence of an explainable model, the cost and time required by the domain experts are high.

The human-level evaluation is similar to the application level in the sense that humans are used to provide feedback. The difference between them is that a wide group of non-expert users are accustomed to grade the human-level results. This method is less expensive and provides a greater number of subjects at the cost of granting less specific feedback. This type of explainability evaluation works well if the only end goal is to provide a good explanation of the output, instead of possibly correcting it. One basic example of this type of evaluation is presenting two different explanations to a user and letting them choose the best one.

Function-level evaluations are mostly used when there is a known metric of the model that is directly tied to explainability. One example applies to decision trees [27, 37]. Since each node represents a condition, a tree with fewer nodes can be said as more explainable. If there are unnecessary nodes or more complex conditions, the model is less explainable. Another example relates to algebraic formulas, where the more complex and nested the operators are, the less explainable the formula is. One metric commonly used is model stability [38], which is defined as the amount that a model's output changes when either the model or its input is perturbed.

3 RELATED WORK

Time series segmentation is often addressed by using a variety of different names. Despite that, it is mostly possible to abstract most methods to a variation of the bottom-up, top-down or sliding window. These algorithms can be used for a plethora of different problems, such as insight extraction or providing model explainability. This chapter aims to explore these uses, in Section 3.1 we summarize other studies in the time series segmentation field in order to understand how segmentation is traditionally employed when the main goal is to create a simpler representation of the data. Section 3.2 will explore other works that use segmentation as a tool to extract data insight and explainability. In Section 3.3 we describe some works with the main objective of bringing explainability to black-box models using segmentation techniques. Section 3.4 presents a comparative table of XTSTree and other related works, highlighting key characteristics for each work addressed.

3.1 Segmentation algorithms

Martí et al. [10] propose a top-down segmentation algorithm with the goal of achieving a low computational cost and easy parameterization while maintaining a decent performance. It applies linear regression on the time series and performs a statistical test to evaluate if the output given by the linear regression and the time series are statistically equal. If they are different, a division is made at the point with the greatest residual error, and the whole process is repeated on the sub-series created. Comparisons between the algorithm and other baselines, among them the top-down, bottom-up and sliding window and bottom-up, showed that it achieves a similar performance while having lower computational cost and simpler parameterization.

Li et al. [39] implement an algorithm to select which segments made using PLR should be maintained. After obtaining the segments through any PLR algorithm, it uses a coefficient of determination R^2 that reflects the reliability of the regression model to represent the time series as a stopping criterion. It calculates R^2 for a segment and the segments created by the cut made at the next depth, accepting the segmentation if at least one of the new segments has a greater coefficient. Redundant segments are also filtered using a minimum length set previously. The authors were able to achieve either better or comparable performance while reducing the number of segments.

Wee & Nayak [40] implement a sliding-window algorithm to reduce a time series to a simpler representation. Elements of the time series are added to the window until the difference between the window's mean and the next element is bigger than a predefined

threshold, at which point a new window is created. Instead of updating the mean whenever adding a new element to the window, it can be updated periodically according to a user-defined value to speed up the reduction process at the expense of accuracy. Although a good reduction is achieved and the algorithm that finds the segment position is similar to Page-Hinkley, comparing the next time series value to the window means can lead to interference from values at the start of a big window.

Keogh et al. [11], in addition to providing an extensive survey on time series segmentation, combines both sliding window and bottom-up algorithms to create a new segmentation algorithm. It initialises a window of size w with enough elements to create about 6 segments using bottom-up. It then applies bottom-up to the window and removes the leftmost segment from it. A process similar to the sliding window algorithm adds new elements to w, and the process repeats itself. The final segments are the set of leftmost segments removed throughout the process.

Fillon & Bartoli [41] propose a new segmentation approach suited for generic datasets based on symbolic regression called Hyper-Volume Error Separation. This approach consists of running a preliminary GP model for Symbolic Regression and locating discontinuity boundaries by analyzing where the greatest errors are given by the model on a dataset. These discontinuity boundaries serve as divisions that segment the dataset into two different partitions, and the previous process is recursively applied until either a user-specified threshold is met or no boundaries are found. Tests made on synthetic time series showed an improvement in final formula accuracy and overall runtime.

3.2 Segmentation for data insight

While there are several studies aiming to divide a time series to optimize the performance of a model or create a simpler representation, there are also plenty of studies in the area of segmentation that aim to find different patterns or behaviours present in the series. These usually view segmentation as a problem of classification. A time series can be segmented by classifying stretches into different classes, resulting in a segmented time series.

One example is Min & Lee [42], where a Temporal Convolution Network creates a latent representation of the time series, which is then sent to a clustering and classification layer that both creates the segments and assigns them to a class. Since these classes are presumed to have similar behaviours, one can create employ the same treatment all segments of the same class instead of having to adopt different treatment for each segment. The classification also helps to understand the patterns present in the series by looking at similarities between segments of the same class.

Matsubara et al. [43] present a similar approach by using a *Hidden Markov Model*

to create *AutoPlait*, an algorithm able to discover and identify many different patterns and change points in a time series and create appropriately labelled segments. The algorithm starts with a single class and segment, and uses a stack structure to store the possible solutions. It then repeatedly pops elements from the stack and tries to split the class and corresponding segment. If the split reduces a cost function based on the currently found classes and segments, both splits and segments are loaded into the stack. Otherwise, they are discarded. AutoPlait was compared with other segmentation algorithms, presenting better results in terms of segmentation and clustering accuracy while not using any user-defined parameter.

One challenge of clustering exclusive to time series is finding clusters based on the structural similarities of data instead of the values or timesteps. Two time series may have identical structures, but different means and variances. In such cases, the same model could be applied to both series. On the other hand, two time series may have similar means and variances but different trends. Therefore, even though they share some similarities, they would be modelled differently. Due to being based on means or differences between values, some segmentation algorithms may not perform adequate segmentation in these situations.

To circumvent this issue, Hallac et al. [21] propose the Toeplitz Inverse Covariance-Based Clustering method for multivariate time series. The method aims to cluster a time series into K different clusters where each cluster is a network of dependencies called *Markov Random Field*, each node of the network is a feature of the series and each edge is a relationship between features. TICC managed to perform better than other baselines. Besides bringing interpretability to a time series by classifying different behaviours, TICC itself is also interpretable since relationships between features are observable.

As previously defined, a time series segmentation task can be defined as grouping adjacent observations in segments, such that observations inside a segment are either similar between themselves or different from observations in other segments. This task involves establishing a similarity between two segments, which is a non-trivial challenge in the context of time series. This is especially important in classification tasks when trying to classify different segments as the same class.

A variety of different methods to establish this similarity have been proposed, such as by Gharghabi et al. [44]. With the goal of creating a domain-agnostic segmentation algorithm, the authors propose the Fast Low-cost Unipotent Semantic Segmentation algorithm (Fluss). It uses an algorithm called STAMP [45] to find the similarity between segments, assuming that every timestep of a time series is the start of a segment. Therefore, for each timestep, it determines where the other segment of the same size that is to it begins. Then, it segments the series based on which timesteps "separate" the most amount of pairs of segments. The segments provide insight into data, since the divisions between

segments are the timesteps where the series is most different, and adjacent segments have different behaviours.

Instead of trying to define the similarity between two series, Chen & Huang [22] implement a segmentation algorithm called TSExplain that leverages the relationship between features and an aggregated metric relevant to the series, such as pack size and total sales in a liquor sales dataset. In this algorithm, the relation between features and the metric is monitored, and segments are based on how well the best m combinations of features correlate to the metric. The authors experimented using three other segmentation algorithms as baselines; Bottom-up, NNSegment [46] and Fluss. TSExplain has surpassed all baselines and was able to effectively explain real datasets while baselines highlighted key features less frequently and created segments with very similar explanations.

Muralidhar et al. [47] propose a segmentation algorithm for multivariate, geographically related time series named *Cut-n-Reveal*. Rather than formulating a similarity between time series, the authors adapted a video segmentation algorithm that finds the segmentation points by representing a timestep t as a combination of previous important timesteps and deriving an affinity matrix between them, creating segments where timesteps are most different. Model explainability is achieved by employing an external model that assigns an explanation vector to each cut containing information on how important each feature is for it. The authors compared their solution to four other segmentation algorithms; *AutoPlait, TICC, DynaMMo* and *Fluss*. The proposed solution performed better segmentations than the baselines in 5 out of 7 datasets, and correctly pointed out the most relevant series for each cut.

3.3 Explaining time series models using segmentation

Several studies aim to bring explainability to other models, and such is not a trivial task, especially in the context of time series. Explainable algorithms, especially in the field of images, heavily rely on visual representations or examples, which is not as easily possible when working with time series.

Sivill & Flach [46] propose an adaptation of the LIME algorithm for image classification algorithms called *LIMESegment*. Given a black-box model for time series classification and a singular time series, the framework's main goal is to grade how important stretches of the time series are concerning the assigned class. The segments are made by iterating two adjacent sliding windows through the series, segmenting where the mean and variance of the windows are different enough according to a threshold. Each segment is perturbed, resulting in a perturbed time series, and a label is given to it by the black-box algorithm. The importance of each segment is given by the weights of a Linear Ridge Regression model that fits the perturbed segments to the assigned label. The LIMESegment

was found to be more faithful and robust than other baselines, faithfulness being defined as how much worse the black-box classifier performs when the most important segment is modified, and robustness being defined as how much different the explanations given are when the time series is perturbed using random noise.

One common method to bring explainability to black-box models is through a feature importance analysis. When applied to time series, these solutions fail to capture changes in feature importance over time. Tonekaboni et al. [33] try to solve this by assigning importance values to each timestep of the series. After a set of features from the time series is selected, the algorithm calculates a metric representing the black-box model's expected drop in accuracy when the last observation is omitted. The same metric is also calculated, except the last value of the time series is replaced by only the selected features instead of removed. The importance value of the set of features at a timestamp is the difference between these metrics, measuring if adding the set of features helps or hinders the model's performance. The author's method either performed similarly or outperformed other baselines while presenting an importance value for each timestep instead of one for the full series.

3.4 Discussion

In this section, we highlight features from each related work and compare them with XTSTree to better distinguish it from other state-of-the-art works. Table 1 presents a comparative table summarizing the main goals and achievements of XTSTree and the related works presented in this section. Column *Study* identifies the work, while column *Most similar segmentation type* shows the most similar segmentation technique to the one used by the algorithm, with the options being the previously explained top-down, bottom-up and sliding window algorithms, and machine learning algorithms. *Machine learning* includes algorithms such as neural networks, Markov models and other learning-based classification techniques. Column *Main goal* presents a short sentence describing the work's main goal to distinguish works with different scopes. *Data insight* and *Algorithm explanation* indicate if the algorithm provides insight into the data's behaviour and if it provides some sort of explainability for the segmentation process, respectively. Finally, *Doesn't require training* depicts if the algorithm requires some kind of training or adaptation before being applied to a time series.

Table 1 – Comparative table between XTSTree and related works.

Study	Most similar segmentation type	Main goal	Data insight	Algorithm explanation	Doesn't require training
XTStree	Top-down	Segment a time series based on its behaviour through an explainable method	[X]	[X]	[X]
Martí et al. [10]	Top-down	Segment a time series efficiently while achieving a low computational cost.	[ ]	[ ]	[X]
Li et al. [39]	Top-down	Optimize the selection of segments made by a top-down algorithm	[ ]	[ ]	[X]
Wee & Nayak [40]	Sliding-window	Segment a time series based on the mean of a sliding window	[X]	[ ]	[X]
Keogh et al. [11]	Sliding-window and Bottom-up	Combine the sliding-window and bottom-up algorithm to take advantage of both algorithm's qualities	[ ]	[ ]	[X]
Fillon & Bartoli [41]	Top-down and Machine learning	Segment a dataset based on discontinuity points given by symbolic regression	[X]	[X]	[ ]
Min & Lee [42]	Machine learning	Cluster and classify elements of a time series to identify hidden behaviours	[X]	[ ]	[ ]
Matsubara et al. [43]	Top-down and Machine learning	Identify and cluster a group of behaviours in a time series	[X]	[ ]	[ ]
Hallac et al. [21]	Sliding-window and Machine learning	Segment a time series and assign segments to predefined classes	[X]	[X]	[ ]
Gharghabi et al. [44]	Bottom-up	Divide elements of a time series where they are most different	[X]	[ ]	[X]
Chen & Huang [22]	Machine learning	Segment a time series based on how features interact with an aggregated metric	[X]	[X]	[ ]
Muralidhar et al. [47]	Machine learning	Create a relation between timesteps and segment a time series where the relation is weakest	[X]	[X]	[ ]
Sivill & Flach [46]	Sliding-window	Explain the behaviour of a black-box model by highlighting relevant stretches of a time series	[ ]	[X]	[X]
Tonekaboni et al. [33]	Machine learning	Calculate feature relevance for a black-box model across time steps	[X]	[X]	[ ]

Several segmentation algorithms resort to some form of linear regression or error-based heuristics to find the best segmentation point, essentially employing a detector for a change in the behaviour of the time series [10, 39, 11]. These solutions may not be the best ones, since there are more robust change detector algorithms. They also do not address explainability for the segmentation process nor data insight, as shown in Table 1. Gharghabi et al. [44] and Wee & Nayak [40] employ segmentation to extract data insight, but both of them fail to provide meaningful explanations for the segmentation method.

Other segmentation algorithms provide good results and achieve data insight and

algorithm explanation by employing sophisticated solutions involving machine learning tools such as neural networks or hidden Markov models [41, 21, 42, 43, 22, 47]. However, this is at the expense of using a more complex and costly learning-based algorithm. Another subarea aims to explain black-box models through segmentation, such as achieved by Tonekaboni et al. [33] and Sivill & Flach [46]. These works can often bring both data insight and model explanation, but they are not stand-alone solutions and rely on a proper black-box model. XTSTree provides data insight and explainability without using similarly complex algorithms, since the change detector and stationarity test it uses do not require training and are simple by themselves.

In general, most studies that aim to provide data insight in the area of time series analysis are seldom related to segmentation [48, 25, 49], and those that are related either fail to provide explainability for their segmentation process or rely on some sort of complex machine learning model. In all studies addressed, XTSTree is the only algorithm able to provide data insight and explainability for its segmentation method while using simpler methods that do not rely on a training process.

4 PROPOSED APPROACH

Our proposed approach is a combination of a change detector and a unit root test that aims to provide insight into the segmentation process. These are implemented through a top-down segmentation framework called XTSTree that simplifies the segmentation process in a segmentation step and a stopping criterion. Although it is possible to implement the segmentation step and stopping criterion as any generic algorithm, XTSTree is specifically designed for a change detector and a stationarity or unit root test. Moreover, the experiments shown in this work use an implementation of XTSTree using a segmentation method based on the Page-Hinkley change detector and a stop condition based on the ADF stationarity test. The following sections will be dedicated to describing the proposed approach and framework implementations.

4.1 XTSTree Framework

The XTSTree's segmentation is recursive: first deciding where to split the data using a decision function, referred to as the segmentation step; and then deciding when to stop splitting by using a stopping criterion that grades the segment. When the segmentation step uses a change detector, the cutting position is the point where a change was detected with the highest possible threshold value, such that only the point with the most amount of change was detected. When using a unit root test, the recursion stops when a stationarity test is applied to the series and the series is sufficiently stationary.

A binary tree stores the cutting points, which can then be retrieved in an ordered manner to create the segments. We chose a binary tree data structure because it treats each sub-series independently for the segmentation step. The binary tree also provides explainability in a similar way to a decision tree. Due to how cuts are made, divisions made closer to the root of the tree signal a more drastic change in the behaviour of the time series, and divisions made closer to the leaves represent less drastic changes in behaviour that are still deemed relevant by the stopping criterion.

Algorithms 4 and 5 shows the recursive segmentation process and creation of the binary tree. Algorithm 4 starts the recursive call of Algorithm 5. Algorithm 5 checks the stopping criterion, and if it is not met, finds the best cutting point, stores it in a new node in the tree, and repeats the described process on the sub-series before and after the cutting point. Figure 10 shows a flowchart of the XTSTree segmentation process.

Algorithm 4: Finds cut positions for given series

```
1  Function find_splits(series) begin
       Data: series: Series to cut
       Result: XTSTree: Binary tree containing cut positions
2      XTSTree ← Node_root
3      XTSTree.root ← find_recursive_splits(series, depth = 0) /* recursively creates the
           tree and returns the root                                              */
4      return XTSTree
5  end
```

Algorithm 5: Recursive function to decide if cuts should be made

```
1  Function find_recursive_splits(series) begin
       Data: series: Series to cut
       Result: Node: Root node containing cut position and children nodes
2      if should stop cutting then return null ;
       /* find best cut according to cut method                                  */
3      cut_position ← find_cut(series)
4      if cut_position = ∅ then return ∅ ;
5      node ← newNode(cut_position)
6      node.right_child ← find_recursive_splits(series[0 : cut_position])
7      node.left_child ← find_recursive_splits(series[cut_position : series.length])
8      return node
9  end
```

4.2 Page-Hinkley and ADF implementation

The stopping criterion we explored uses the unit root test Augmented Dickey-Fuller (ADF) test to determine if a series is non-stationary to stop the segmentation process. We use a version of the ADF test that does not take the series' seasonality into account, meaning that a series that is non-stationary due to being seasonal will be classified as stationary. The intuition behind this is that along a change detector as a segmentation process, series that are stationary will not yield any cuts. Therefore, when the ADF tests signal that the series is stationary, it either is stationary, and the segmentation process will not produce any cuts, or it is non-stationary, but there are seasonal patterns inside it, and the segmentation process will separate these patterns into subseries. If a series is stationary, and after being cut it is non-stationary, we interpret that it had seasonal patterns contained in it that now were successfully isolated.

In order to provide explainability for the cuts, the ADF test result is stored as an indicator of how close the segmentation process is to stopping. Therefore, it is possible to monitor how each cut changes this value, indicating how effective it is. Algorithm 6 describes the ADF stopping criterion. If the series has the minimum length required by the ADF test, the test values for the series are calculated. Then, the difference between the specified stopping value and the calculated p-value is returned as the test score, and the segmentation stops if the score is positive.

We suggest an implementation for the segmentation method that uses the Page-Hinkley change detector to find the point with the most significant change in the series' behaviour. To achieve this, the threshold value for the detector is chosen such that only one

Algorithm 6: Augmented Dickey-Fuller test stopping criterion function

```
1 Function adf_stop_condition(series; stop_value) begin
      Data: series: Series to be checked
      Input: stop_value: Minimum critical value required to stop cuts
      Result: test_confidence: The more positive the value, the more confidence that the series is
              non-stationary
2     if series is too small then return 0 ;
3     test_confidence ← adfuller(series) − stop_value
4     return test_confidence
5 end
```

change is detected for the whole series. The method finds the best Page-Hinkley threshold value by performing a binary search, reducing the threshold value if no cuts are found on the series, and increasing it if more than one cut is found. The cutting process stops either after a threshold that produces only one cut is found, or after a specified number of iterations have passed. If the cutting process stopped due to exceeding the iteration limit, then for each cut found by the biggest threshold value, the stopping criterion is applied to both subseries created by it. The cut that created the pair of subseries that have the greatest combined score is chosen. If there is not a threshold that produces at least one cut, then the series is not cut at all. Algorithm 7 describes the implementation for the described method.

4.3 XTSTree Example

To better illustrate the proposed solution, we present an application example where it is desired to create a SR model for a time series with different behaviours, greatly benefiting from a segmentation using XTSTree. In this example, as well as throughout the rest of the work, SR is leveraged as a powerful tool to enhance explainability. Symbolic regression provides a concise mathematical representation of a numerical sequence, thereby offering improved interpretability. This formal mathematical transcription not only gives a clear understanding of the underlying patterns but also facilitates a more comprehensive explanation of the time series data. The series used in the example, presented in Figure 11, is a combination of several sinusoidal-like series with added noise, as shown below.

Algorithm 7: Page-Hinkley segmentation method

```
1  Function find_page_hinkley_cut(series; starting_threshold, max_number_iterations, delta)
   begin
       Data: series: Series to be cut
       Input: starting_threshold: Starting value for the page-hinkley threshold
       Input: max_number_iterations: Maximum number of iterations before making a decision
              on the cut
       Input: delta: Delta parameter for the Page-Hinkley change detector
       Result: cut_position: Position that yields the best cut
2      min_threshold ← 0
3      max_threshold ← −1
4      threshold ← starting_threshold
5      for iteration in max_number_iterations do
6          pagehinkley ← newPageHinkley(threshold, delta)
7          initiate cuts as empty lists
8          for element in series do
9              update pagehinkley with element
10             if drift detected by pagehinkley then  adds index of element to cuts ;
11         end
12         if cuts.length = 1 then  return cut[0] index ;
13         if cuts.length > 1 then
14             min_threshold ← threshold
15             if max_threshold is negative then
16                 threshold ← threshold + threshold/2
17             else
18                 threshold ← (max_threshold − threshold)/2
19             end
20         end
21         if cuts.length = 0 then
22             max_threshold ← threshold
23             threshold ← (threshold − min_threshold)/2
24         end
25     end
26     if no cuts found after max_number_iterations then  return null ;
27     if cuts.length > 1 after max_number_iterations then
28         pagehinkley ← newPageHinkley(min_threshold, delta)
29         initiate cuts as empty lists for element in series do
30             update pagehinkley with element
31             if drift detected by pagehinkley then  adds index of element to cuts ;
32         end
33         final_cut ← first_cut_in_cuts
34         max_stat ← stop_function(series[0 : cut]) + stop_function(series[cut : series.length])
35         if stat > max_stat then
36             max_stat ← stat
37             final_cut ← cut
38         end
39     end
40     return final_cut
41 end
```

$$f(x) = \begin{cases} 5 - 0.05x + \sin x/20 + \epsilon, & \text{if } x \leq 100 \\ \sin x/20 + \epsilon, & \text{if } x \geq 100 \text{ and } x \leq 300 \\ 0.03x - 10 + \sin x/20 + \epsilon, & \text{if } x \geq 300 \text{ and } x \leq 450 \\ 50 - 0.1x + \sin x/20 + \epsilon, & \text{if } x \geq 450 \text{ and } x \leq 500 \\ \sin x/20 + \epsilon, & \text{if } x \geq 500 \text{ and } x \leq 700 \\ 0.03x - 23.3 + \sin x/20 + \epsilon, & \text{if } x \geq 700 \text{ and } x \leq 850 \\ 90 - 0.1x + \sin x/20 + \epsilon, & \text{if } x \geq 850 \text{ and } x \leq 900 \\ \sin x/20 + \epsilon, & \text{if } x \geq 900 \text{ and } x \leq 1100 \\ 0.03x - 36.7 + \sin x/20 + \epsilon, & \text{if } x \geq 1100 \text{ and } x \leq 1250 \\ 130 - 0.1x + \sin x/20 + \epsilon, & \text{if } x \geq 1250 \text{ and } x \leq 1300 \end{cases}, \epsilon \in \{-0.2, 0.2\}$$

The first step of our example is to apply Genetic Programming for SR over the whole series, without any segmentation, to extract a formula that can represent the entire series. Figure 12 shows SR results along with the actual series. The presented model was the one that yielded the best accuracy. As can be seen, the formula found can barely describe the series' behaviour. The formula should ideally reproduce not only the more prominent periodic behaviour seen every 400 steps, but also the smaller repetitions every 100 steps. Aiming to get better results from SR, we now employ XTSTree to cut the series, with ADF-test as the stopping criterion and Page-Hinkley as the cut method. The outcome is presented in Figure 13.

XTSTree subdivided the original time series into 8 sub-series, which have a way simpler behaviour. The colours used in the series help us understand the reasoning behind the cuts. The darker the series' colour, the more it has deviated from the initial behaviour according to the Page-Hinkley change detector, and when it reaches the threshold, the series is cut. With the series now divided into stationary segments, SR can be applied to each of them. Figure 14 depicts these results.

Employing SR on the stationary segments yielded a set of formulas that describe the series significantly better than before, as can be seen in Figure 14. Moreover, if the initial series was part of a seasonal time series, such as in temperature or energy consumption over a week, the extracted formulas could be applied to future time steps, as seen in Figure 15. In this figure, the same pattern from the initial time series is repeated but using a different seed for random components and concatenating both series. Then, the formulas given by the leaf models were reapplied for the newly created series. As shown in Figure 15, the values obtained from the formulas are visually well-adjusted to the new series.

Figure 16 provides an overview of the XTSTree, illustrating the segmented series and associated formulas. It offers a comprehensive view of the entire XTSTree generated from the time series example. The tree structure follows a binary tree formation with six depth levels. At the root level, the complete time series is depicted. Moving to the first layer of depth, we observe the split generated by the most divisive sample, resulting in two nodes without any leaves. This indicates a point where a change in the time series pattern occurs but does not present leaves capable of representing the expected level of simplification. Each split was designed to avoid complex formulas by merging segments with completely different behaviours.

In the second layer, we can observe three leaves and a node. Each leaf displays an optimized formula obtained through SR. The corresponding nodes represent large windows from the original time series, where it is not possible to produce the expected level of explanation according to the experimental setup. Layers 3 to 5 illustrate a sequence of multiple splits identifying small segments represented by simple formulas based on sine and

hyperbolic sine functions. These formulas typically consist of less than five trigonometric operators.

As we delve deeper into the layers, more complex formulations become apparent, but they remain less complex than the complete formula. XTSTree effectively represents each segment as a simple formula. Moreover, the XTSTree's structure can be utilised to assess the time series' complexity, serving as a meta-feature in the observed domain.

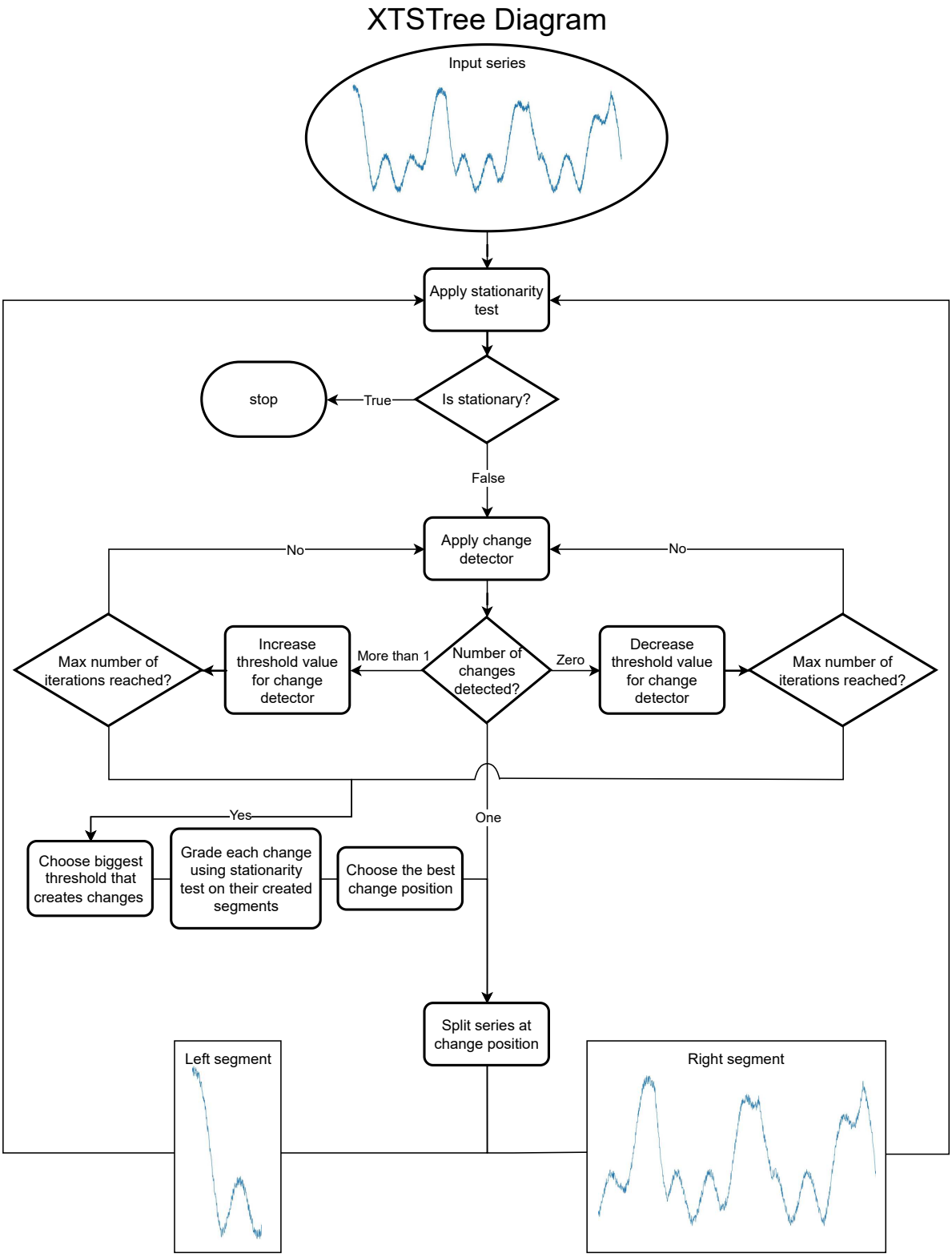


Figure 10 – XTSTree's flowchart.

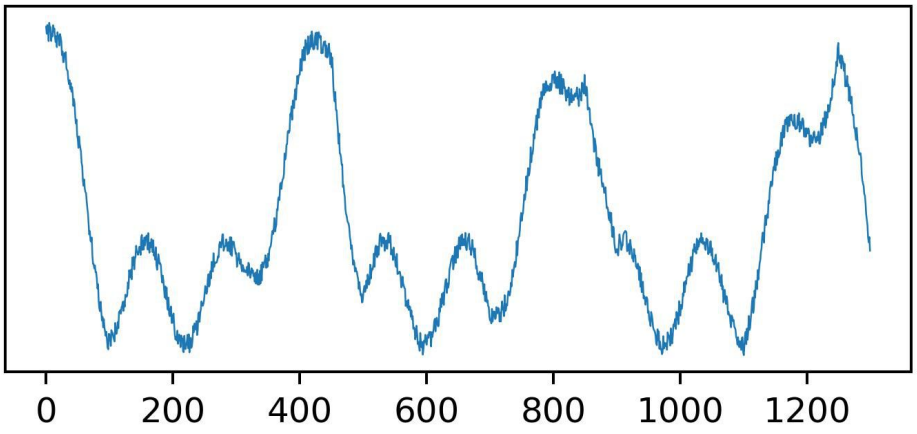


Figure 11 – Time series created using a combination of several sinusoidal-like series with added noise.

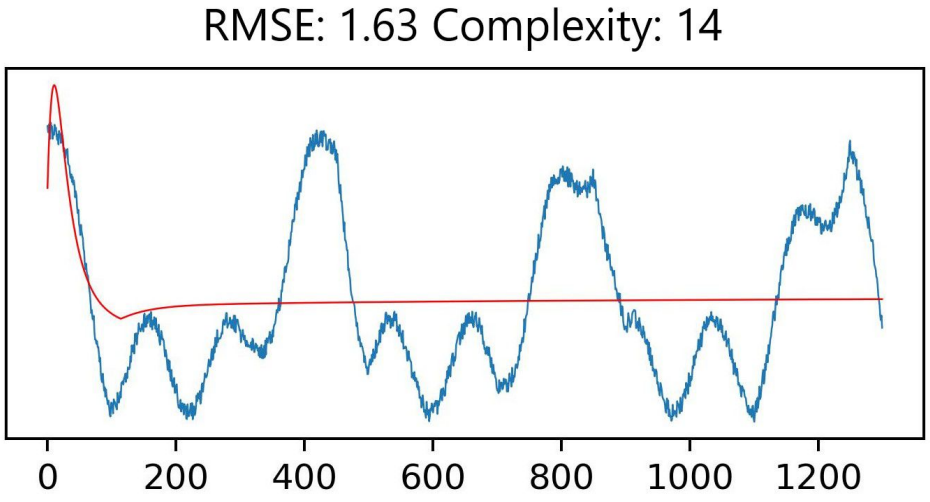


Figure 12 – Symbolic Regression applied using accuracy for formula selection. RMSE was computed for the formula against the complete series. Complexity is a combination of number of operators and how nested they are in the formula, as computed by PySR [1].

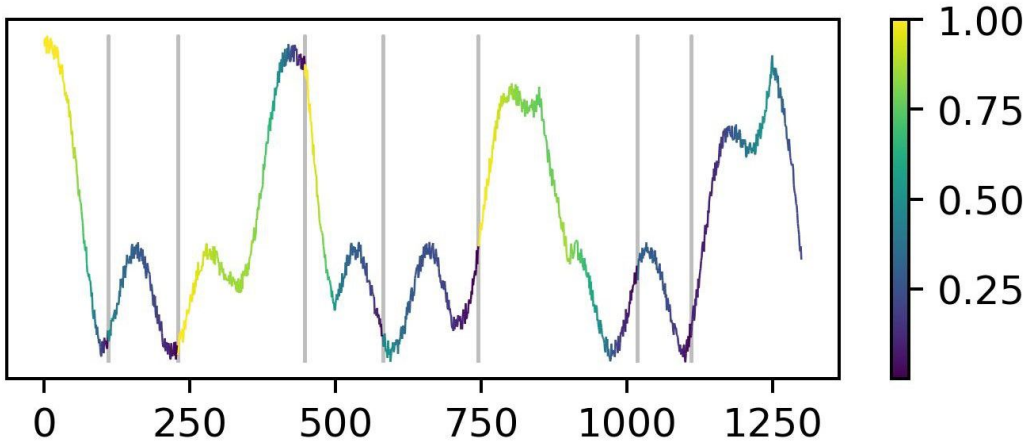


Figure 13 – Time series used in the example after the cuts performed by XTSTree. The colour scale ranges from 0 to 1.

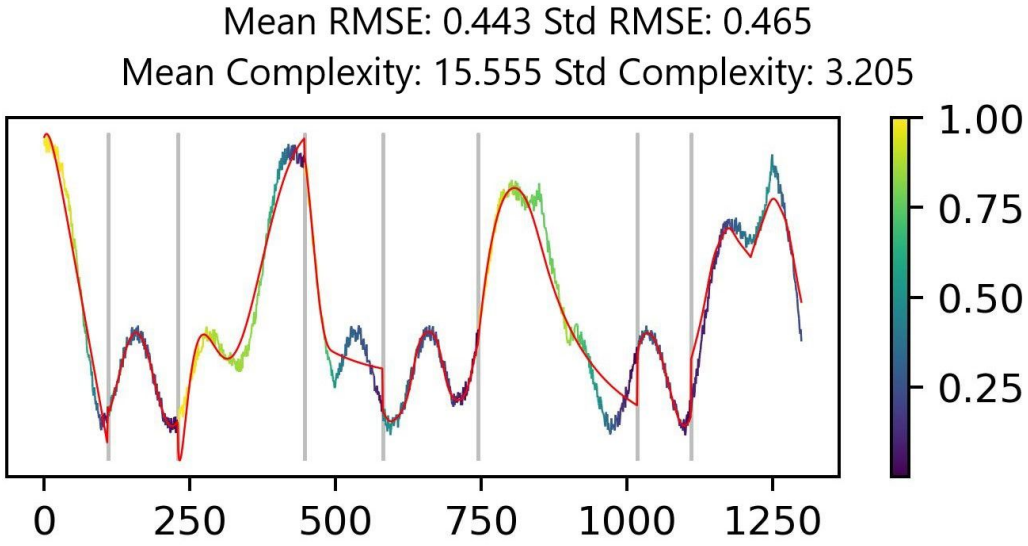


Figure 14 – Symbolic Regression applied on the resulting leaves. colour scale ranges from 0 to 1. RMSE was computed for each formula against the corresponding sub-series. Complexity is a combination of the number of operators and how nested they are in each formula, as computed by PySR [1]. In both metrics, the mean and standard deviation were taken for each leaf.

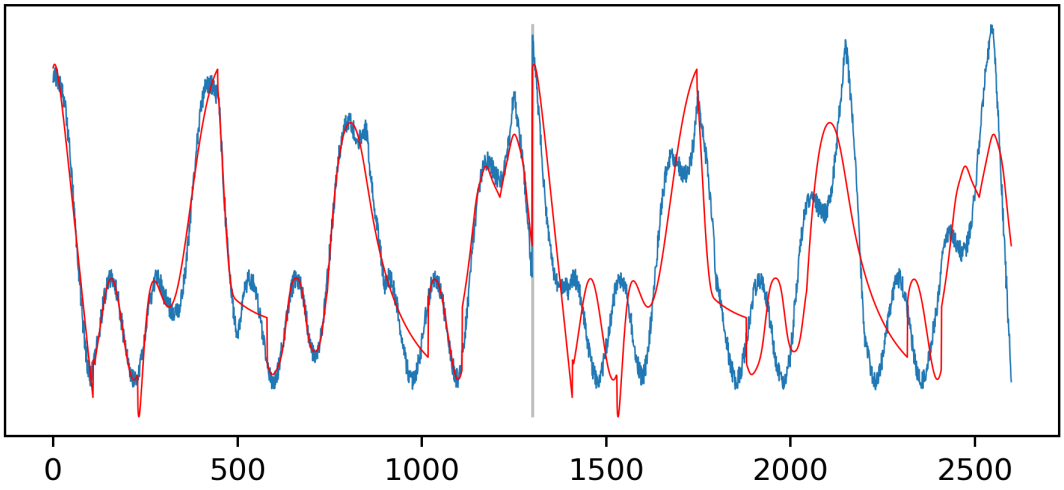


Figure 15 – The same series shown in fig. 11 concatenated with another series created using the same method and another random seed. The red line represents the formulas shown on fig. 14 and applied to the new series.

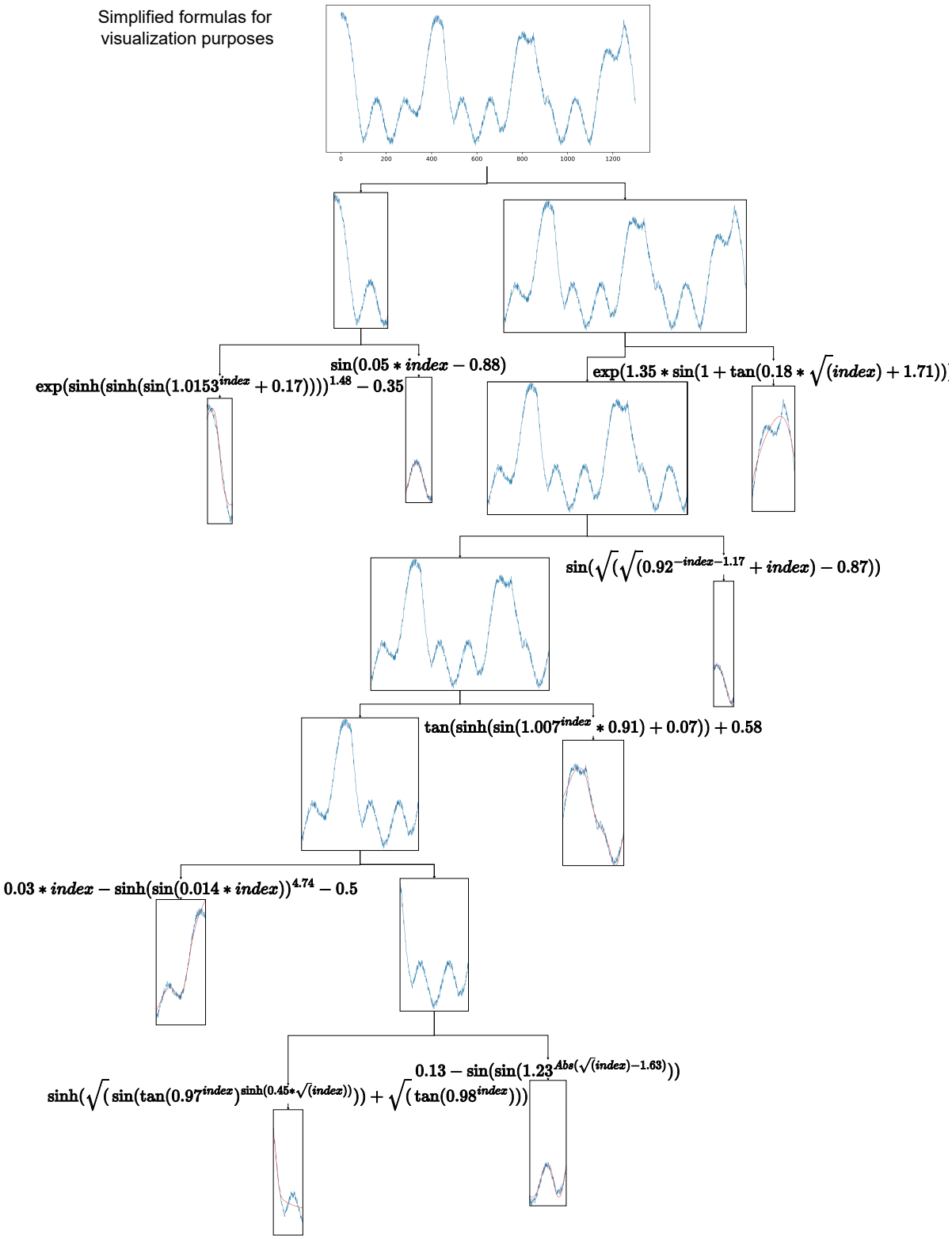


Figure 16 – Representation of the XTSTree obtained from an example Time Series using **ADF** and **Page-Hinkley**.

5 EVALUATION AND RESULTS

We conducted experimental studies in order to evaluate the capabilities of our proposed solution. Two tests were made, first by comparing XTSTree using the Page-Hinkley segmentation method and ADF stopping criterion against other two additional segmentation methods and one additional stopping criterion serving as baselines. Our second experiment aims to compare XTSTree using Page-Hinkley and ADF with two implementations of the classic top-down segmentation algorithm. Finally, we present a case study on a real-time series to exemplify how XTSTree provides data insight as well as explainability.

5.1 Comparison with XTSTree baselines

Our first experiment aims to guarantee that the Page-Hinkley segmentation method and ADF stopping criterion are effective for the segmentation problem. Since the main goal of XTSTree is not to perform some kind of error reduction, but to provide data insight and explainability, we compare our method against a set of naive solutions to check if our method can outperform them. We implemented two baseline segmentation steps for comparison where cuts are made randomly and at the middle of the series. As for the baseline stopping criterion, we implemented one based on the depth of the tree. All three baselines are used with the XTSTree framework.

5.1.1 Baselines implementation

The Depth stopping criterion is based on the tree's depth and acts as our baseline for the stopping criterion since we can set multiple depths as stopping points for testing purposes. Two baseline segmentation methods were implemented: one creating cuts periodically at the middle of the series, and another creating cuts at random positions. These segmentation methods address different biases. Periodic cuts produce predictable subseries of the same size, whereas Random cuts may create sub-series that are bigger or smaller than needed, or break patterns by chance. These stopping criteria and segmentation methods were combined with the Page-Hinkley and ADF, creating a total of six different models using each segmentation method (Page-Hinkley, Periodic, and Random) and stopping criterion (ADF test and Depth).

In total, six models were compared, these being combinations of different methods for the segmentation steps of XTSTree with the main goal of validating the individual variations.

5.1.2 Datasets

We carried out the experiments using real data that represents humidity data collected by the Rural Development Institute of Paraná (IDR-PR), Brazil. Readings were collected by a sensor every fifteen minutes starting from January 2015 up to November 2021, totalling 96 samples per day. These readings were then split into segments of 5, 10, 15 and 20 days, resulting in 747 univariate time series of four different lengths (480, 960, 1440 and 1920 readings).

These time series were chosen because they fit the expected scenario for XTSTree: seasonal time series that change over time, have seasonal patterns and can benefit from explanations about their behaviour. Table 2 briefly describes these datasets, and Figure 17 shows one of the series used.

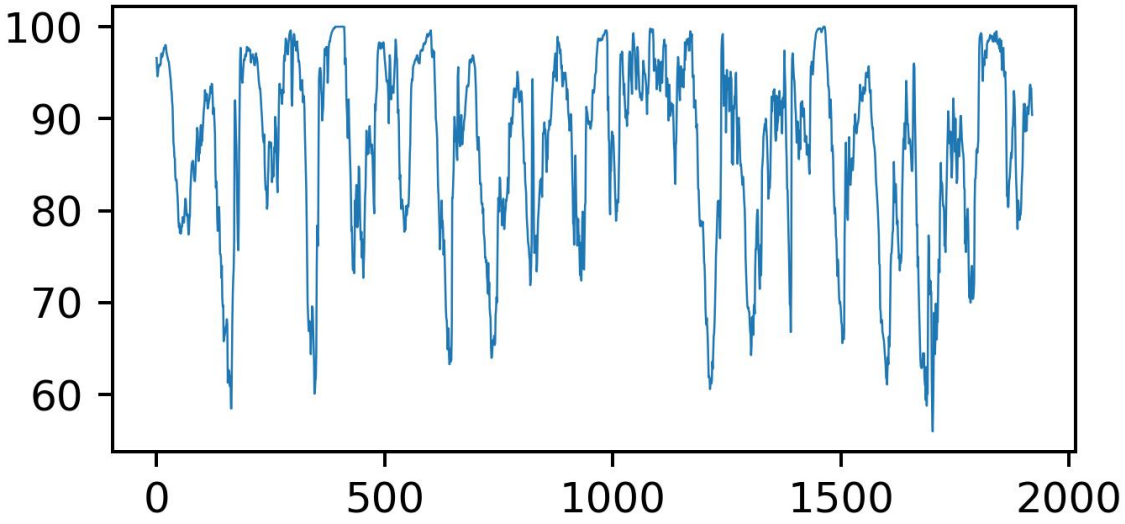


Figure 17 – Series of twenty days containing information on air humidity.

Table 2 – Meta-data of the time series analysed in the experiments

Days	Length	# Time Series
5	480	358
10	960	179
15	1440	120
20	1920	90

5.1.3 Evaluation

To compare the quality of the six XTSTree versions and showcase the potential for explainability, we used the SR algorithm presented in Cranmer [1]. First, we modelled the original time series with SR using the timestamp as a feature and the value as the

prediction target. Then the series is segmented and the signals on the XTSTree leaves are modelled using the same parameters as before. In our experiments, we utilized the PySR [1] and Julia [50] implementations of SR. We selected both because they are high-performance platforms for implementing SR algorithms, which can be particularly useful for time series data.

The hyperparameters chosen for the Page-Hinkley segmentation method were the standard parameters, these being *delta*: 0.005, *alpha*: 0.999, min number of iterations: 100, and other methods require no parameters. We chose a stop value of 0.05 for the *ADF* stopping criterion, meaning that the test will label a series as stationary correctly with 95% confidence, and a depth value of 3 for the Depth stopping criterion. As for the PySR algorithm, 10 iterations, 20 populations and a population size of 40 were used. These parameters refer to the genetic algorithm and are the implementation standard. Two model selection options were leveraged, 'accuracy' and 'best'. The former chooses the model based on model accuracy, while the latter also takes into account the model complexity. The possible operators chosen were the standard for PySr, being '+', '-', '*', '/' and 'pow' for binary operators, representing the four basic math operations and exponentiation. As for unary operators, we chose negative, exponential, absolute, logarithmic, square root, sine, tangent, hyperbolic sine and sign. The XTSTree implementation can be accessed in this public repository[1].

5.1.4 Accuracy Improvement

The length of a time series is an important factor when trying to create a model for it. As a time series grows in length, so does its complexity. However, identifying the proper segmentation point in a series to best improve its representation is a tricky task. To assess the XTSTree's capacity to improve model accuracy, we compared the error obtained for the whole signal (original series) and the mean of the models' error obtained from the signal of each leaf. We used Root Mean Square Error (RMSE) to measure the error, defined by the formula:

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}{n}}.$$

where y_t is the value of Y at time t, $\hat{y}_t$ is the predicted value of Y at time t and n is Y's length. In our experiment, $\hat{Y}$ is obtained by giving the timestamps as input to the best model given by PySR.

Figure 18 shows the RMSE according to the series' length. The red and blue dots represent the RMSE for SR models applied on the whole series and for models applied on XTSTree with ADF as the stopping criterion, respectively. The lines represent the average RMSE for both methods. The results show the contribution of the ADF test as

[1] github.com/BobVitorBob/XTSTree

the stopping criterion, improving the XTSTree's scalability. As presented in Fig. 18, the average RMSE for models applied using XTSTree remains constant regardless of the series length, whereas the ones that were applied on the whole series exhibit increased error as the series length increases.

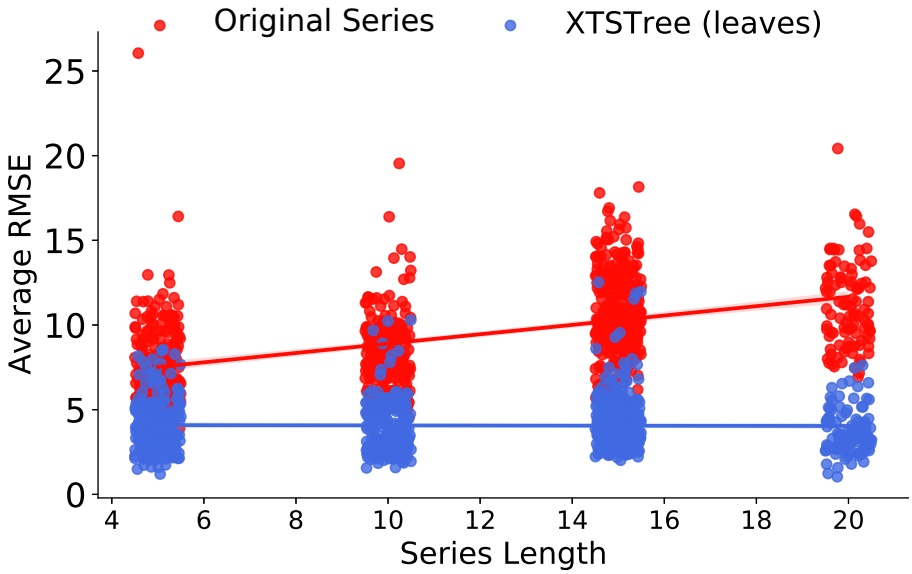


Figure 18 – RMSE by series' length in days, Symbolic Regression on XTSTree leaves using ADF test as stopping criterion. Red dots represent formulas fitted to the original series while blue dots represent average RMSE for formulas fitted on the XTSTree leaves.

To highlight the significance of ADF as a stopping criterion, we carried out further experiments using different stopping criteria. A fixed depth of three levels was used as the stopping criterion for XTSTree. Figure 19 presents the same graph as Figure 18, but with grey dots representing models relying on XTSTrees with depth as the stopping criterion. Although XTSTree obtained more accurate models in comparison to the original time series, both results show a similar error-increasing pattern proportional to the original signal length. This is due to the number of cuts being fixed to a maximum depth of three. The bigger the series is, the greater the number of cuts it generally needs. Should the depth value be increased, the XTSTree performance would be increased on lengthier series, but too many cuts would be made for smaller series. Moreover, this depth parameter would need to be chosen for each series length, while the ADF test addresses well all series sizes. This demonstrates the benefits of using the ADF test as the stopping criterion to improve both the accuracy and scalability of XTSTree.

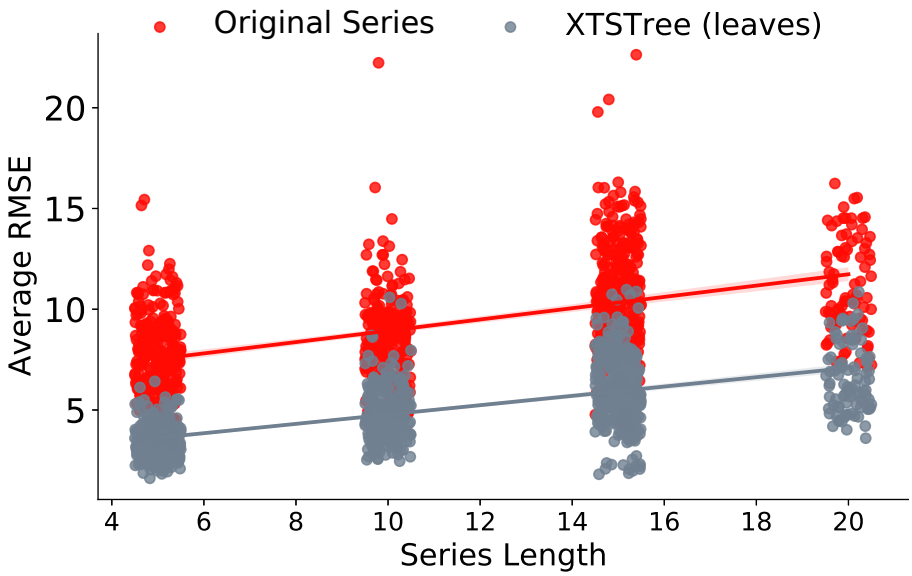


Figure 19 – RMSE by series' length in days, Symbolic Regression on XTSTree leaves using a fixed depth of 3 as stopping criterion. Red dots represent formulas fitted to the original series while grey dots represent average RMSE for formulas fitted on the XTSTree leaves.

We evaluated different segmentation methods for XTSTree. To determine if there are any significant differences between using Periodic, Random, or Page-Hinkley as the segmentation method, we conducted a statistical analysis based on the non-parametric Friedman test over 140 time series. We used the post hoc Nemenyi test to infer which differences are statistically significant. To find statistical superiority in terms of RMSE reduction, we compared the three segmentation methods using ADF as the stopping criteria. Figure 20 shows the Nemenyi post hoc test results on the obtained error (RMSE), where we can observe that Random and PH obtained the most accurate models, with no significant differences between them, and the Periodic segmentation methods performed the worst.

It is important to mention that the number of splits and the tree depth obtained from the Random segmentation method was bigger than Page-Hinkley, which means that even though Page-Hinkley and Random are equal in terms of accuracy improvement, Page-Hinkley is better because it produces fewer leaves, which means fewer segments to model using SR. Page-Hinkley also has the benefit of being consistent due to being deterministic, and also explainable since cuts are made intentionally.

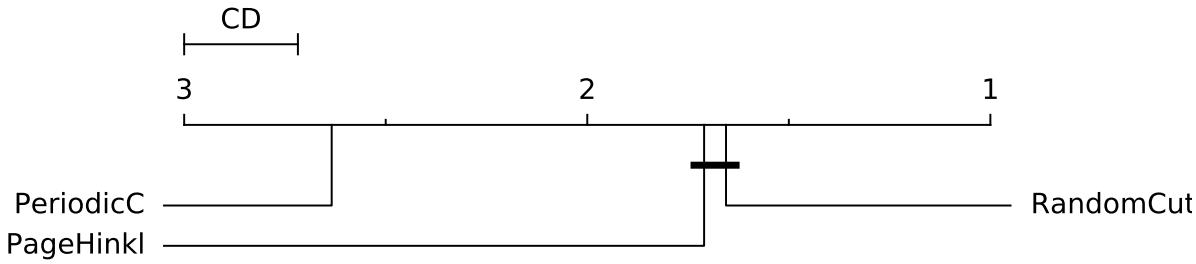


Figure 20 – Critical Distance diagram based on the Nemenyi post hoc test using CD of 0.282 considering the RMSE of three segmentation methods (Periodic, Random, and PH) with 140 paired time series.

5.1.5 Complexity Reduction Analysis

We performed an analysis of complexity reduction based on default PySR hyper-parameters for inducing SR. We define complexity reduction as the difference between the complexity of the formula, as calculated by [1], for the original series and the average complexity of the leaves' formulas. The XTSTrees had all three different segmentation methods and used ADF as the stopping criterion. Figure 21 shows a positive difference when using PH as a segmentation method for all series lengths. It is important to mention that the highest reduction of complexity was observed for series with 15 days of signal acquisition. The random segmentation method was able to deliver slight improvements within 5 days of signals and remarkable results with 10 and 15 days. However, when processing 20 days, the random selection of a split point led to more complex equations. The periodic segmentation method presented a slight improvement with 10 days, but for other signal lengths, it resulted in more complex representations than the whole signal.

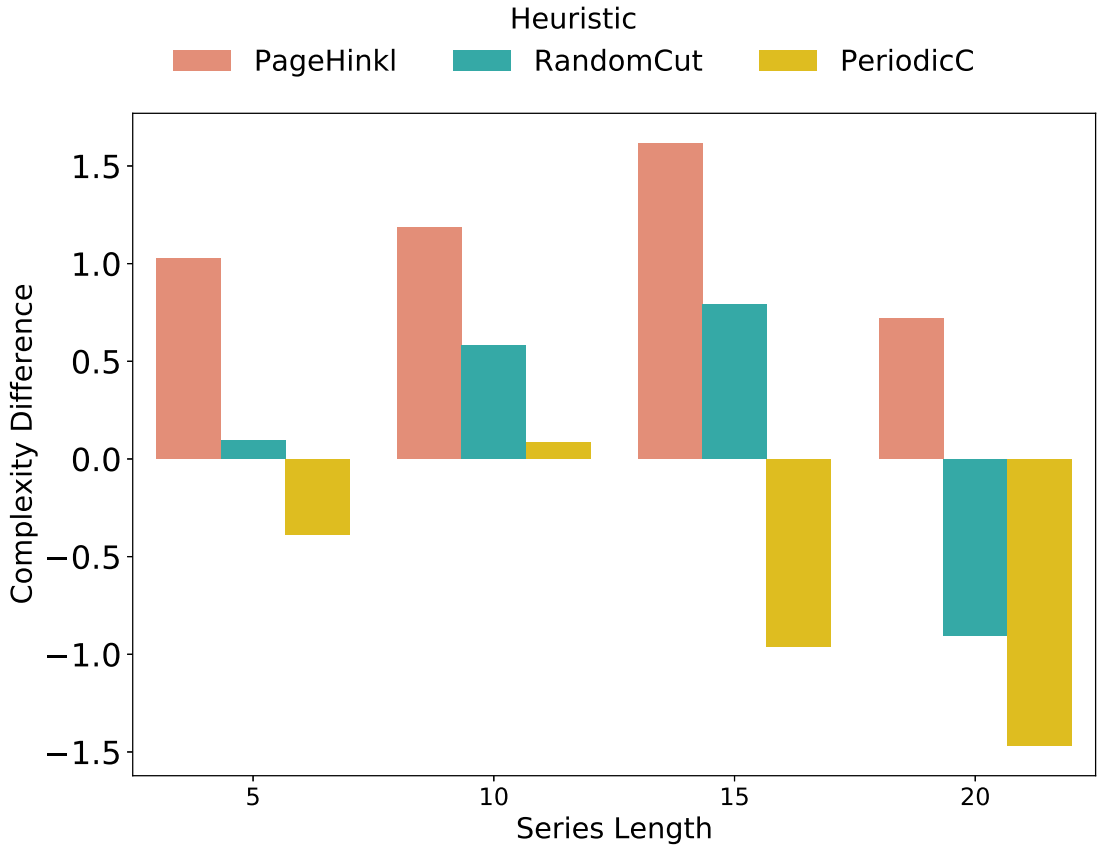


Figure 21 – Average complexity reduction using three segmentation methods (PH, Random, and Periodic) with ADF stopping criterion.

To verify the statistical validity of the complexity reduction, we followed the same statistical test that was previously employed (Friedman and Nemenyi post hoc test). Figure 22 shows the Nemenyi post hoc test results on the obtained complexity analysis. In the figure, it is possible to observe that all results were statistically different, with PH being the most promising method for reducing complexity, followed by Random and Periodic, respectively

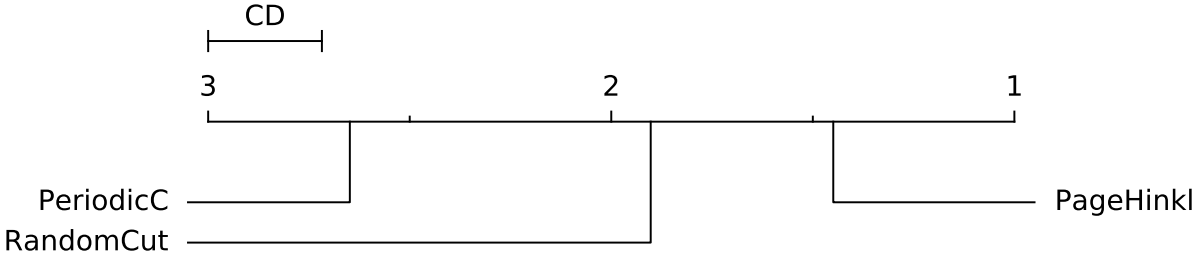


Figure 22 – Critical Distance diagram using CD of 0.282 based on the Nemenyi post hoc test considering the difference of complexity between XTSTree and original series using three segmentation methods (Periodic, Random, and PH) with 140 paired time series.

We considered that interpretability could also be evaluated in terms of the XTSTree number of splits, i.e., the number of recommended segmentations on a time series. Using the same experimental setup, we observed that the Periodic segmentation method generates fewer segments driven by the hyperparameters employed. Conversely, Random as a segmentation method generates more cuts without reaching the improvements in terms of the reduction of complexity provided by PH. Finally, the PH segmentation method proved to deliver the best trade-off, improving the accuracy and reducing complexity when using XTSTree to process a time series. Figure 23 presents the relation between the number of segments (cuts) with various time series lengths with the different segmentation methods.

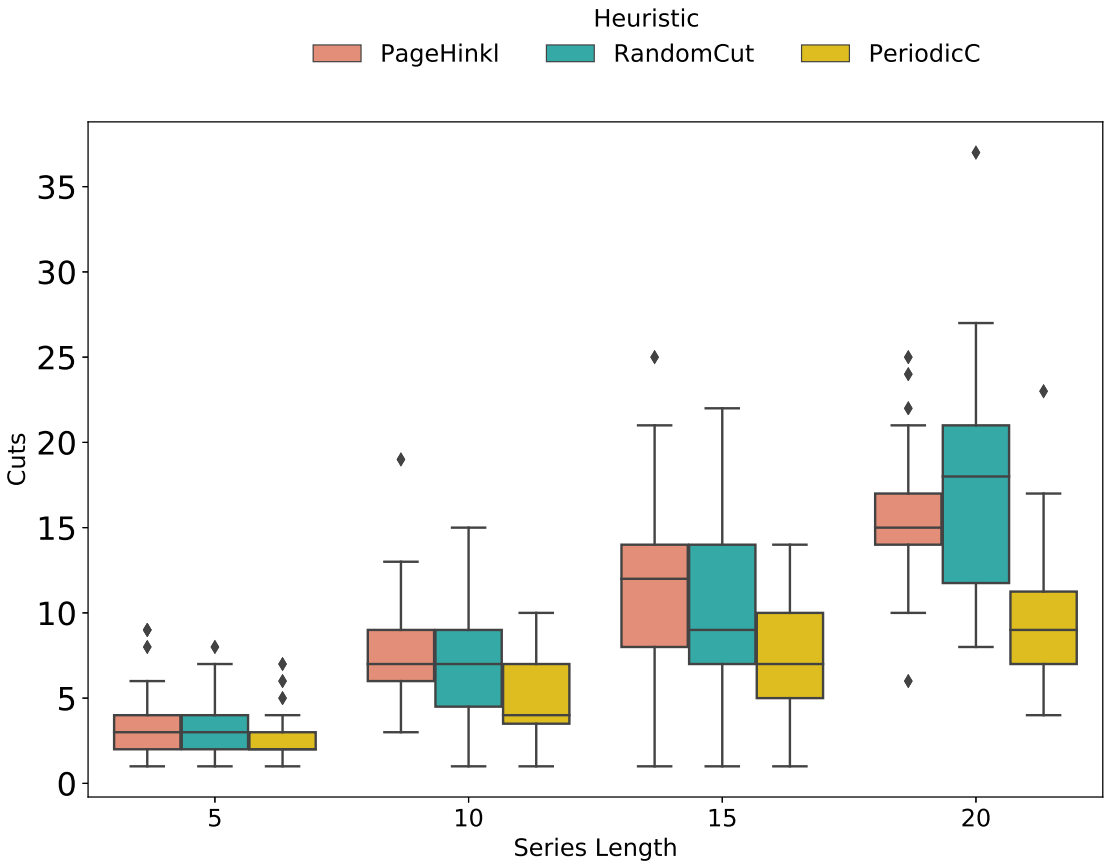


Figure 23 – Number of segments (cuts) with various time series lengths for different segmentation methods.

5.1.6 Computation Time Comparisons

To study how XTSTree affects computation time, we investigated several scenarios and observed that the impact is mainly related to the final size of the tree. The time cost of using XTSTree was negligible during our experiments when compared to the time spent on SR, hovering around less than 1 s. When using the max depth stopping criterion, there was a small increase in the time cost of modelling the original series. However, when growing an XTSTree using the ADF stopping criterion, deeper trees are created, more

splits are processed, and more time is spent on modelling. Figure 24 shows the difference obtained using ADF and Max Depth of 3 levels. Experiments were made on an I5-1035G1 with 16 GB RAM.

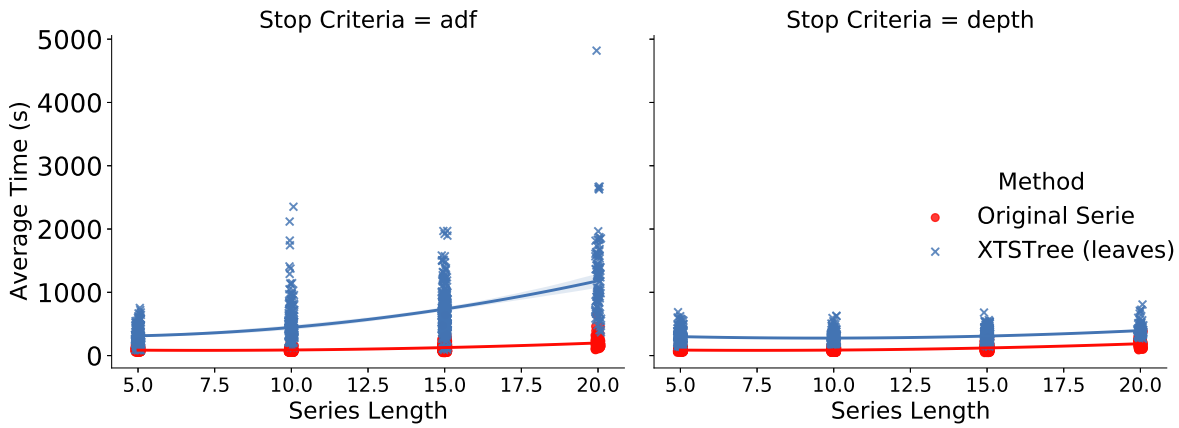


Figure 24 – Average time consumption by length of series in days with ADF and Depth as stopping criterion.

Figure 25 shows the Nemenyi post hoc test results on the obtained time. We investigated the influence of the different segmentation methods. Due to the nature of the Periodic method, it was the fastest one to create the cuts, and since it creates less leaves, it was also the fastest to apply SR. The Random method provided deep trees, but the computational cost of this criterion is very low. Finally, PH provided average depth trees but is the slowest to create splits. The statistical evaluation based on the Friedman and Nemenyi post hoc tests shows that there are no methods statistically similar, with Periodic being the fastest to create splits, and Page-Hinkley the slowest.

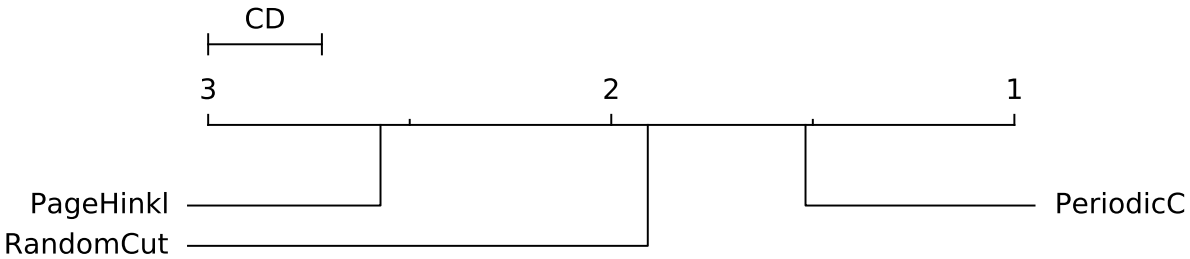


Figure 25 – Critical Distance diagram using CD of 0.282 based on the Nemenyi post hoc test considering the time of three segmentation methods (PeriodicC, RandomCut, and PageHinkl) with 140 paired time series.

5.1.7 Entropy evaluation

One way to evaluate the effectiveness of the segmentation step is to compare how much it contributes to reaching the stopping criterion. Should two methods have the same performance in other areas, the segmentation that reaches the stopping criterion

more efficiently is better. This can be done by computing the difference between grades given by the stopping criterion before and after a cut. The smaller this grade gets, the closer the algorithm is to stopping. Therefore, an algorithm that can reduce this grade more efficiently is also one that creates better cuts. Moreover, since these values are stored with the cut position, it is possible to understand which cuts are more important for the segmentation, bringing a level of explainability to the tree.

This experiment was made with the same datasets described in Table 2. We applied the same models of the experiment described previously. For each tree, the grade given by the ADF stopping criterion was stored for each subseries. Then, at each cut, the difference between the grade before and after the cut was calculated. We then calculated the average value for all cuts of each tree, yielding a mean entropy reduction grade for each tree. The results can be seen in Figure 26, where each bar represents the mean reduction and standard deviation.

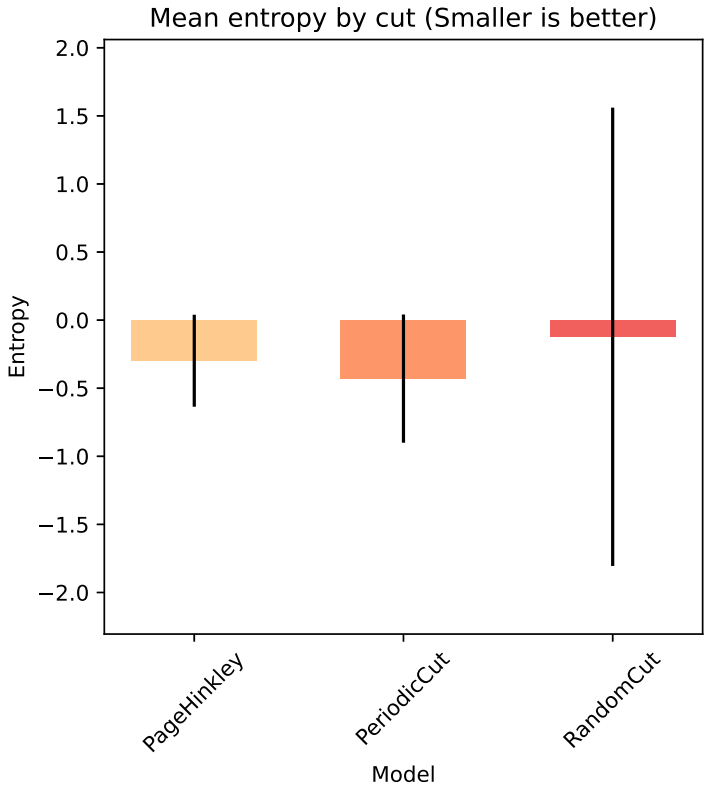


Figure 26 – Entropy comparison between segmentation methods. It represents the mean approach towards the stop condition, with a more negative value representing a greater approach.

Figure 26 shows us that the random cut performed the worst by far due to its random behaviour, with a high variation and almost zero mean reduction per cut. Periodic cut performed best, but as seen in the complexity reduction analysis, while it is more efficient at reaching the stopping criterion, its cuts provide a worse complexity reduction

than the other methods. Page-Hinkley provides a balance between segment quality and cutting efficiency.

5.2 Comparison with top-down algorithms

Intending to understand how XTSTree compares to the classic top-down models from literature, we implemented two versions of the top-down algorithm. Both implementations follow the top-down method described in the background section of [11], using linear regression to find cutting positions and stop the process. The difference between them is that one uses the index of the series as input, while the other uses a lag window containing the series' values.

5.2.1 Top-down implementation

The two variations of the top-down segmentation algorithm implemented follow closely the description in [11]. Both variations are based on linear regression, creating a model for each segment and segmenting the series at the position where the regression error was the highest. Segments are created until the root mean squared error of the model reaches a predefined threshold. The first variation uses the series' timesteps as input, while the other uses a lag window of a predefined size. To choose an appropriate lag, window sizes of 4, 24, 48 and 96, representing 1, 6, 12 and 24 hours respectively were tested in a preliminary test. A window size of 48 was chosen since it had the best trade-off between execution time and performance.

The predefined thresholds used for both variations were chosen between 25%, 50% and 75% of the RMSE given by the linear model fitted to the full series after a run of preliminary tests. The chosen thresholds for the timestep and lag variation were 25% and 50% of the original error, respectively, since the greater threshold values resulted in almost no cuts made. A value of 50% of the original RMSE for the index-based often required only one cut due to how the algorithm works.

5.2.2 Datasets

Our experiments for this section were made using the same data described in Section 5.1, but with a reduced number of series due to the extended execution time of the top-down algorithm. The reduced number of time series was due to an extended execution time of the index-based top-down algorithm. In series containing 20 days, some executions took as long as 8 hours. In contrast, the other methods finished the same series in as little as 10 minutes. This is mainly due to the number of segments created by the index-based top-down algorithm, and it will be discussed in the evaluation subsection. To better compare XTSTree and lag-based top-down using a greater number of time se-

ries, we decided to stop the index-based top-down execution. Comparisons between all three algorithms used the datasets described in Table 3. and a more extensive comparison between XTSTree and lag-based top-down was made with the datasets described in Table 4.

Table 3 – Meta-data of the time series used by all three models

Days	Length	# Time Series
5	480	47
10	960	7
15	1440	9
20	1920	8

Table 4 – Meta-data of the time series used only by Page-Hinkley and Lag-based top-down

Days	Length	# Time Series
5	480	340
10	960	168
15	1440	112
20	1920	81

It is worth noting that some segments created by all algorithms failed to execute the symbolic regression. All series that could not run on all available models were discarded, and all series described were successfully executed with all models. In some situations, XTSTree decided to not segment the series due to it already being non-stationary. Therefore, two evaluations were made, with and without series where no cuts were made.

5.2.3 Evaluation

We segmented each series from the dataset using the three models, applying symbolic regression to each segment evaluating MAE, RMSE, execution time, number of segments, mean segment complexity and standard deviation of segment complexity. Although similar to the previous experiment, the parameters for the symbolic regression were different. We used the 'best' model selection criteria, 10 populations with a population size of 30, and 40 iterations, due to providing similar performance with a faster execution time. For the binary operators, we used the four basic math operations and exponentiation, represented by '+', '-', '*', '/' and 'pow'. The unary operators used were 'sqrt', 'sin', representing square root and sine, respectively.

This change in hyperparameters was made after experimentation with the PySR tool. We realized a relation between iteration size and formula complexity, where a small number of iterations resulted in a smaller general complexity. We also noticed that several

operators were either redundant or too specific, making the final result worse. These changes led to an overall increase in computation time, which is why we opted to use only the 'best' selection criteria. We note that preliminary tests were made using both criteria, and the results between the models tested were similar. Figures 27 to 30 show the comparison between the models using data described in Table 3.

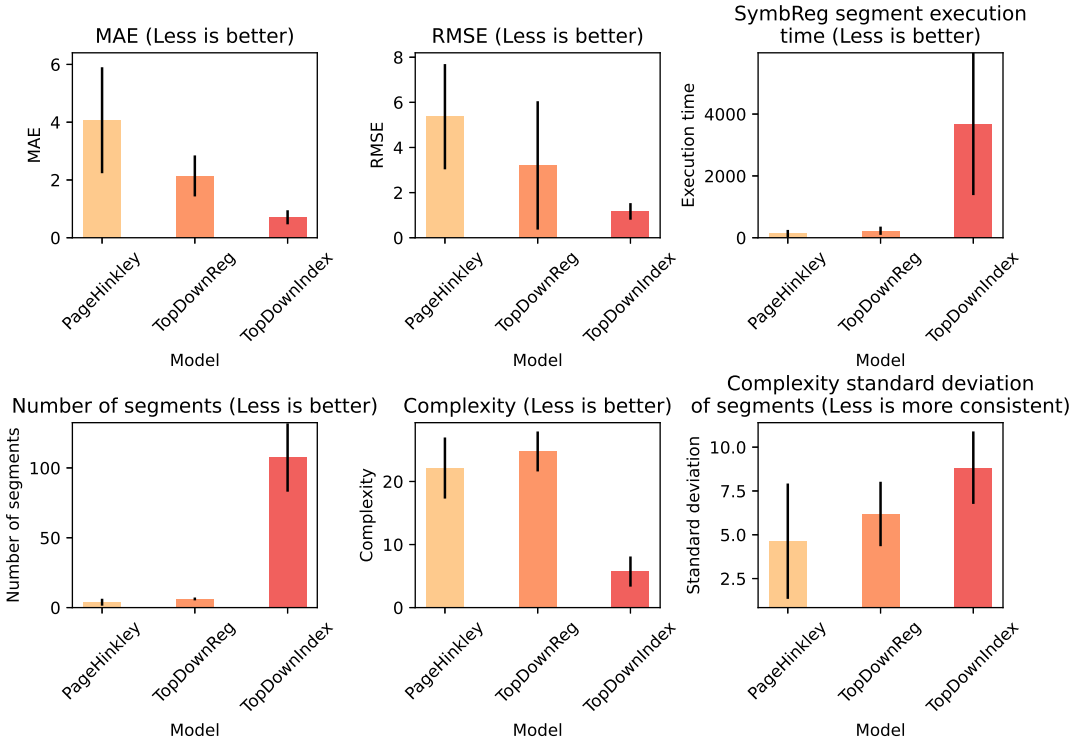


Figure 27 – Metric comparison between XTSTree (PageHinkley), Lag-based Top-down (TopDownReg) and Index-based Top-down (TopDownIndex) for time series with a length of five days.

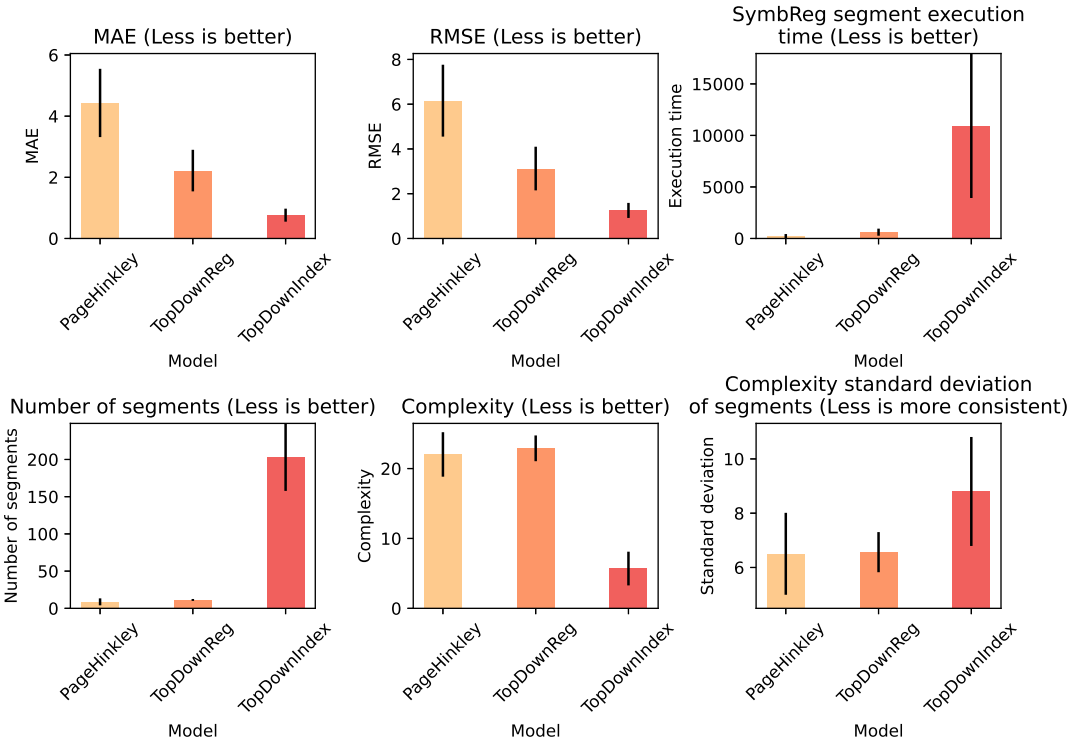


Figure 28 – Metric comparison between XTSTree (PageHinkley), Lag-based Top-down (TopDownReg) and Index-based Top-down (TopDownIndex) for time series with a length of ten days.

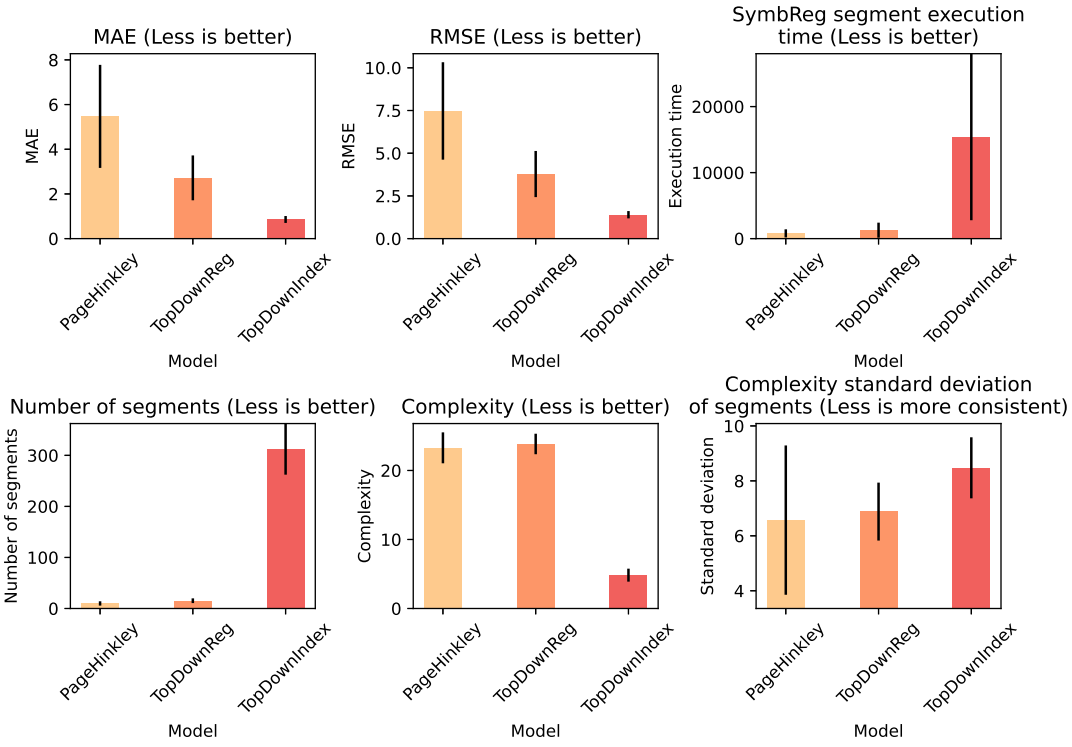


Figure 29 – Metric comparison between XTSTree (PageHinkley), Lag-based Top-down (TopDownReg) and Index-based Top-down (TopDownIndex) for time series with a length of fifteen days.

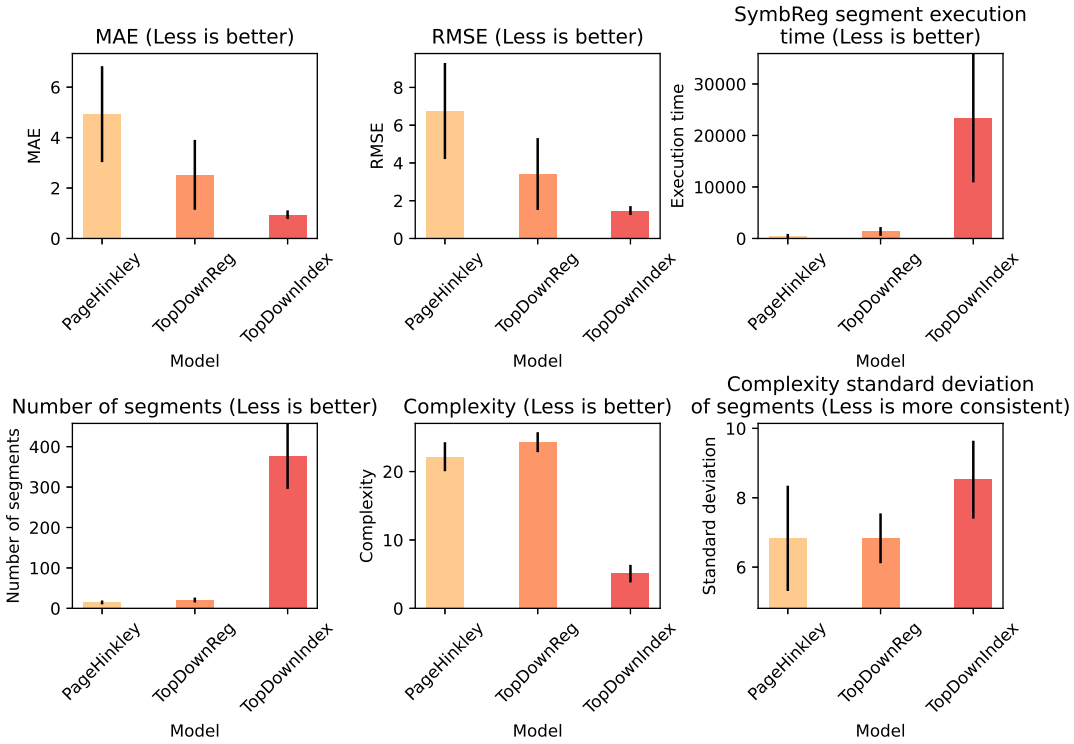


Figure 30 – Metric comparison between XTSTree (PageHinkley), Lag-based Top-down (TopDownReg) and Index-based Top-down (TopDownIndex) for time series with a length of twenty days.

The Index-based Top-down algorithm performed better in terms of MAE and RMSE. However, this is due to the number of segments it creates for even the smallest series, an average of more than one hundred segments. These segments naturally are very small, making it very easy to create formulas for them. This is not a positive outcome, because even for a series of nearly five hundred items, it takes about one hour while the other two models take less than three minutes, not justifying the reduction in error. Moreover, these small segments hold very little value in terms of data insight. To continue with a more meaningful result analysis, we present the results obtained using only XTSTree and Lag-based Top-down in Figures 31 to 34.

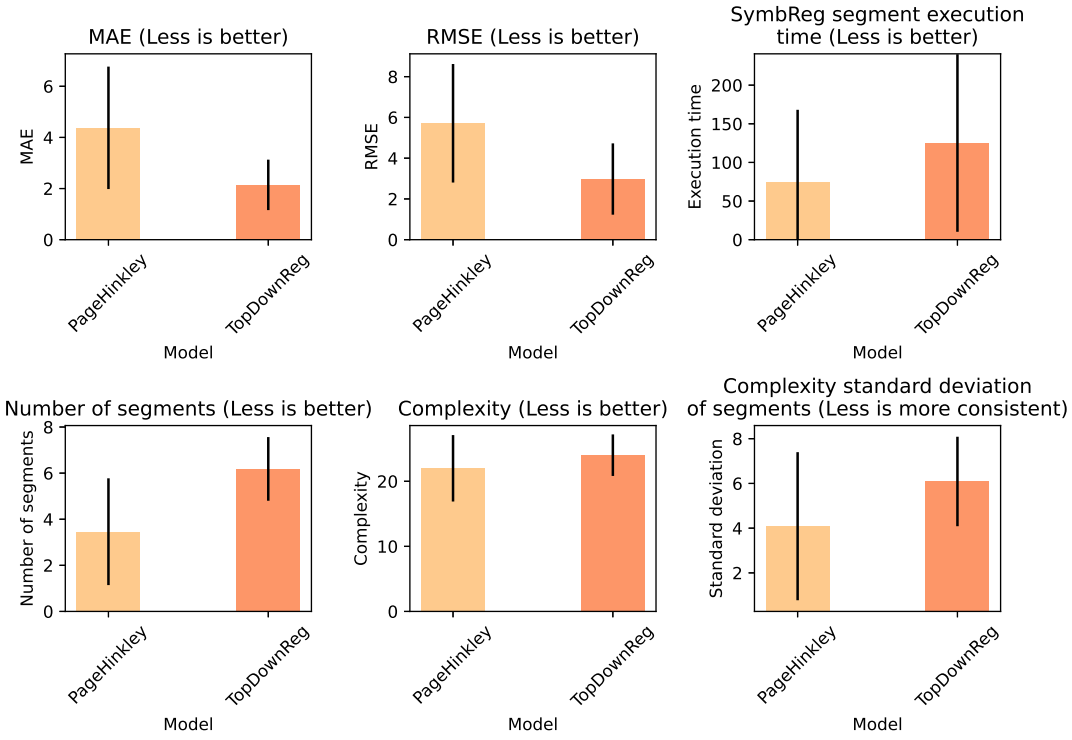


Figure 31 – Metric comparison between XTSTree (PageHinkley) and Lag-based Top-down (TopDownReg) for time series with a length of five days.

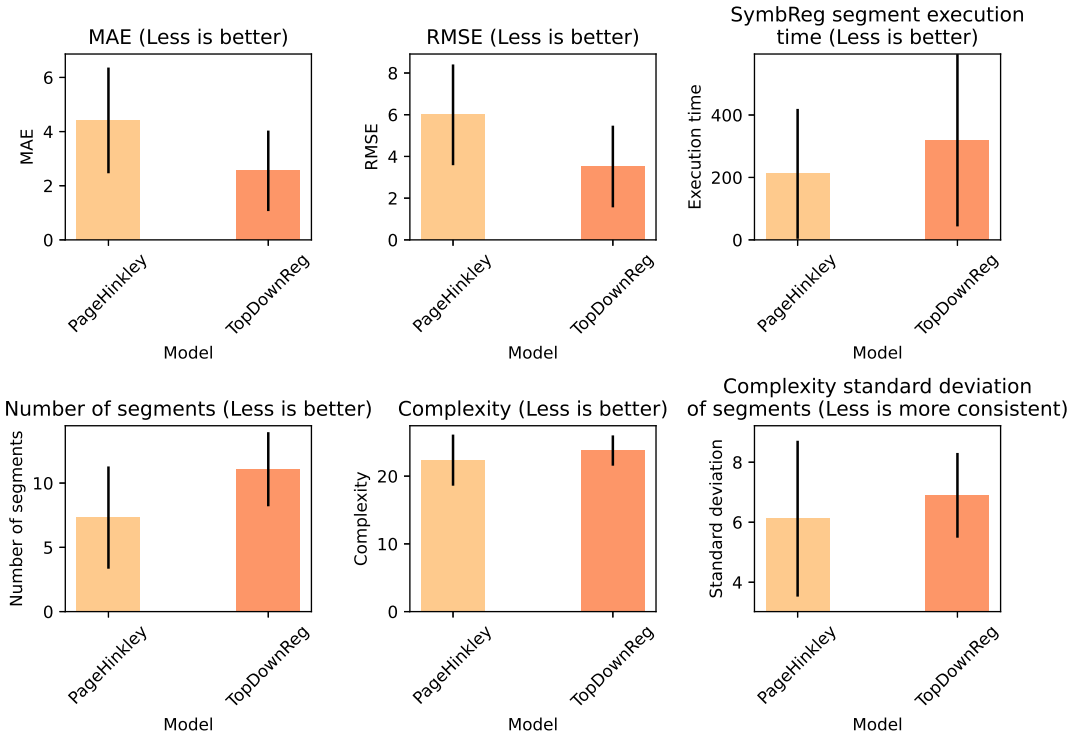


Figure 32 – Metric comparison between XTSTree (PageHinkley) and Lag-based Top-down (TopDownReg) for time series with a length of ten days.

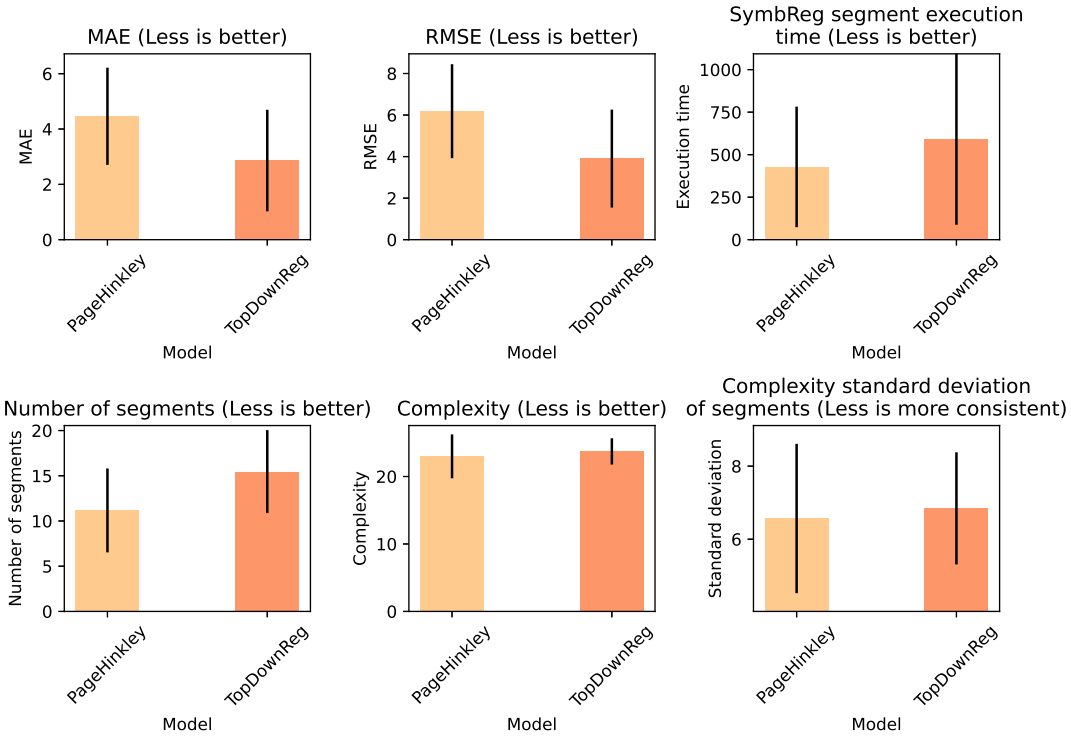


Figure 33 – Metric comparison between XTSTree (PageHinkley) and Lag-based Top-down (TopDownReg) for time series with a length of fifteen days.

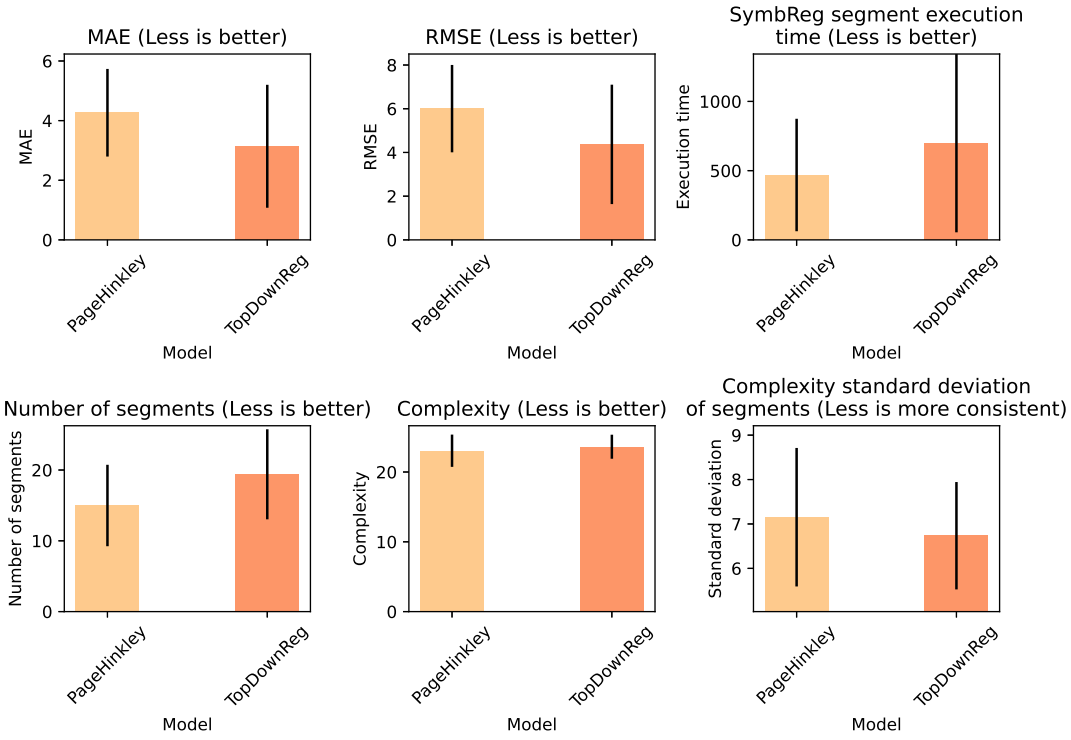


Figure 34 – Metric comparison between XTSTree (PageHinkley) and Lag-based Top-down (TopDownReg) for time series with a length of twenty days.

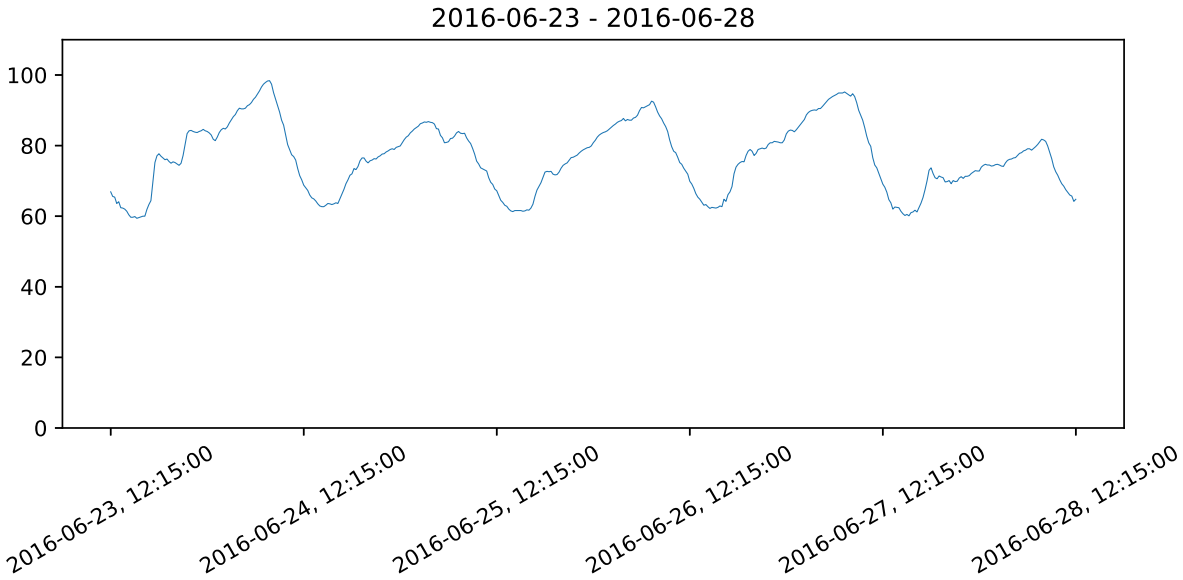
We observe that the results are similar across all series' sizes. XTSTree performed

worse regarding MAE and RMSE, although by a small margin. It performed marginally better than the Top-down in terms of complexity, number of segments and execution time. We note that the difference between the number of segments and execution time is proportional to both error values, meaning that both algorithms performed similarly, with XTSTree creating fewer cuts at the expense of accuracy.

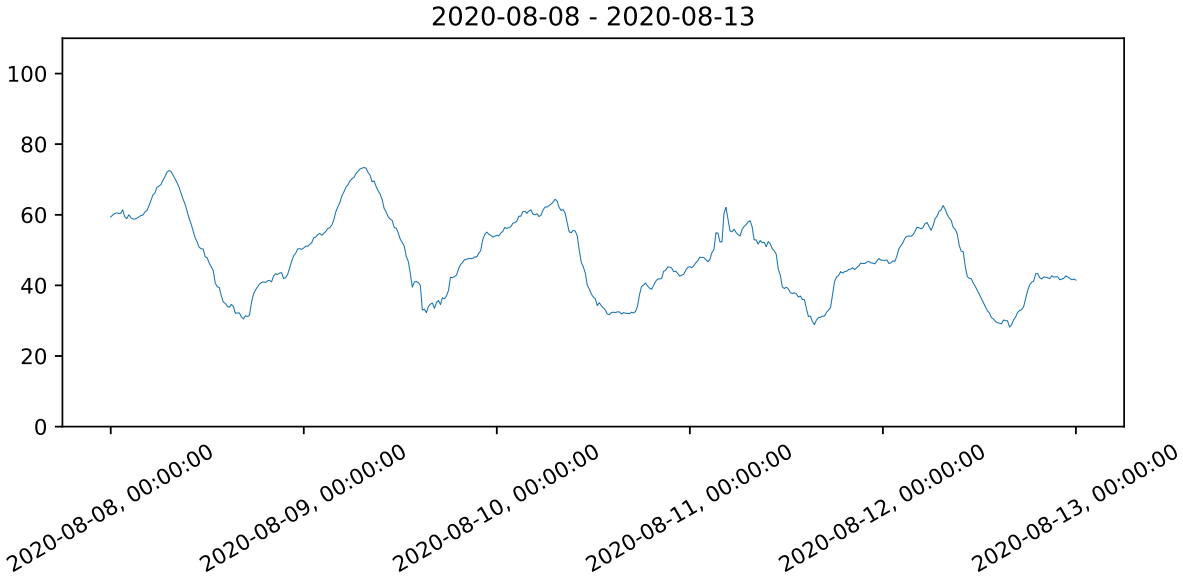
It is important to note that by analyzing the evaluated metrics, there is no meaningful distinction between top-down and XTSTree, instead there is a trade-off between the number of segments and error. However, XTSTree provides value through a tree structure that offers data insight while maintaining explainability for the segmentation method. Moreover, XTSTree addresses the task of detecting a change in behaviour using a change detector. These distinctions favour the use of XTSTree for the segmentation task over the Top-down algorithms.

5.3 Case study on explainability

This section presents a case study where XTSTree can gather data insight on real time series provided by the Institute of Rural Development of Paraná (IDR-PR). The main goal is to exemplify how XTSTree provides data insight in a real context, while also highlighting which of its features provide explainability on the segmentation process and data insight. Figure 35 shows two different time series containing information for air humidity over five days. It is visually noticeable that these series have periodic behaviours with a mean duration of about a day.



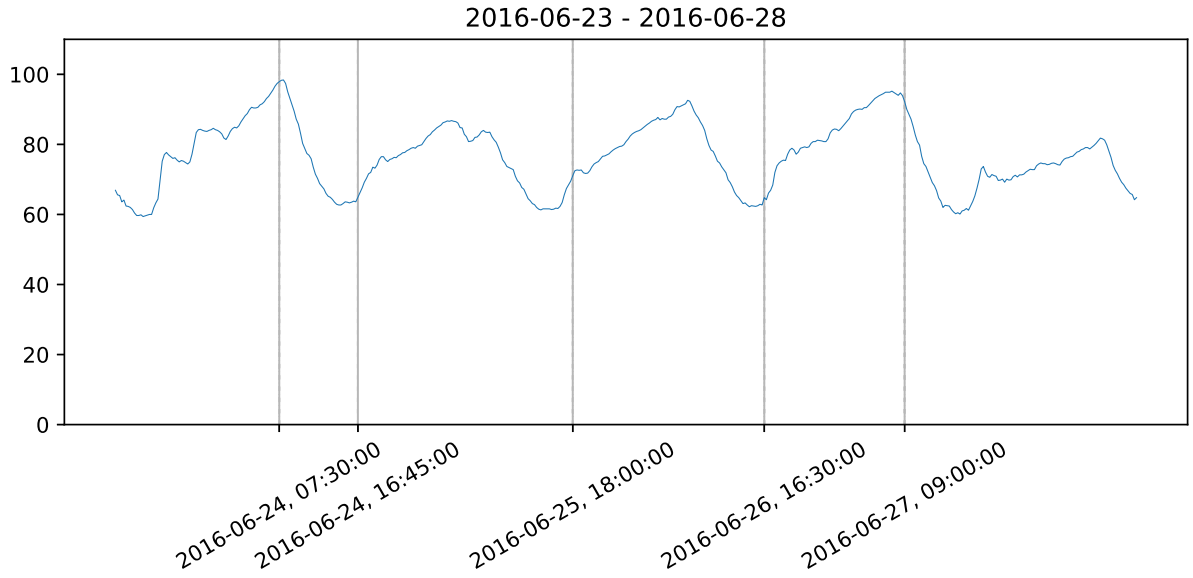
(a) Series on air humidity between 2016-06-23, 12:15:00 and 2016-06-28, 12:15:00 with readings collected every 15 minutes.



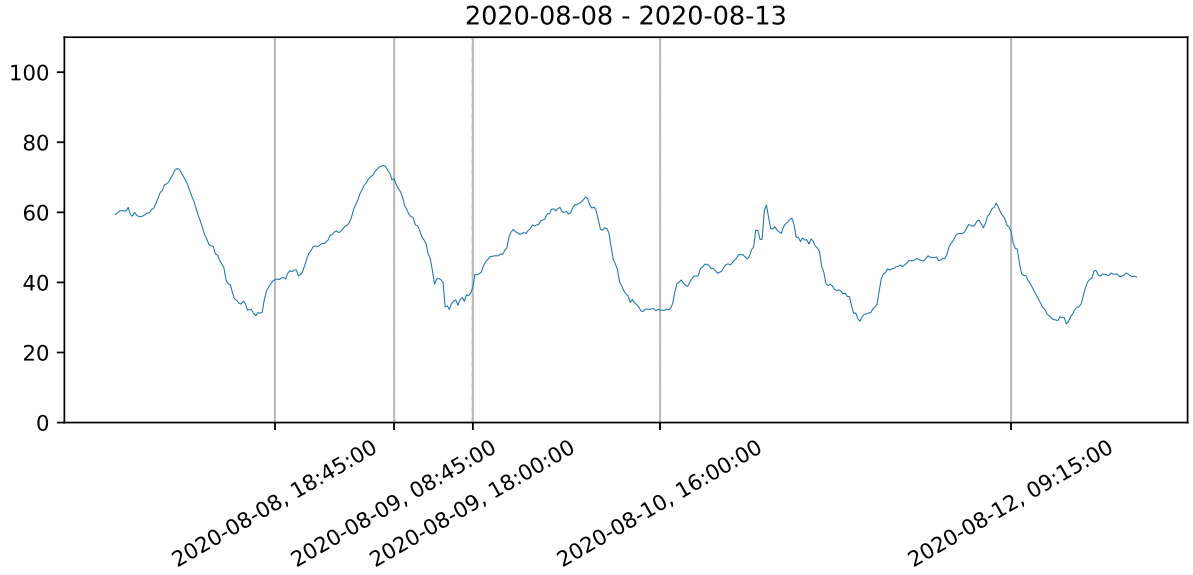
(b) Series on air humidity between 2020-08-08, 00:00:00 and 2020-08-13, 00:00:00 with readings collected every 15 minutes.

Figure 35 – Two time series measuring air humidity over five days.

As an analyst, it may be of interest to divide the series into smaller series with behaviours that are easier to analyse. We can apply XTSTree to both series, yielding the structures depicted in Figure 37. These structures represent the binary tree utilized by XTSTree to store its cuts. The time series on which the PH segmentation method was applied, the selected cut positions and stop condition values are also depicted. At the top of each series is the result of the stop condition, with a number greater than 0 meaning that the series requires a segmentation. Figure 36 shows the complete series with all segments made, represented by the grey lines.

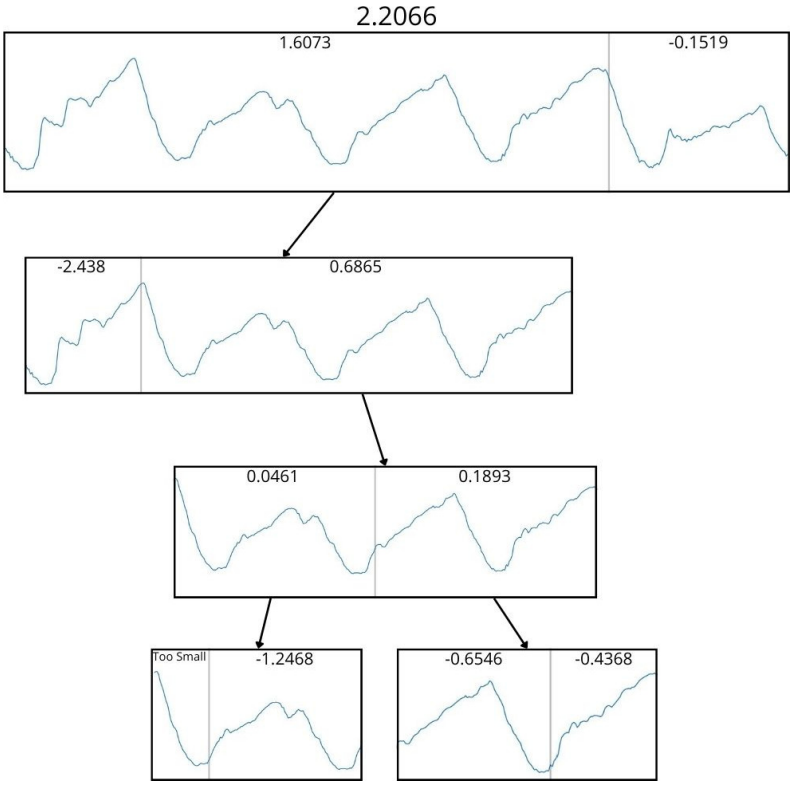


(a) Time series measuring air humidity over five days. Grey lines represent segmentation points created by XTSTree.

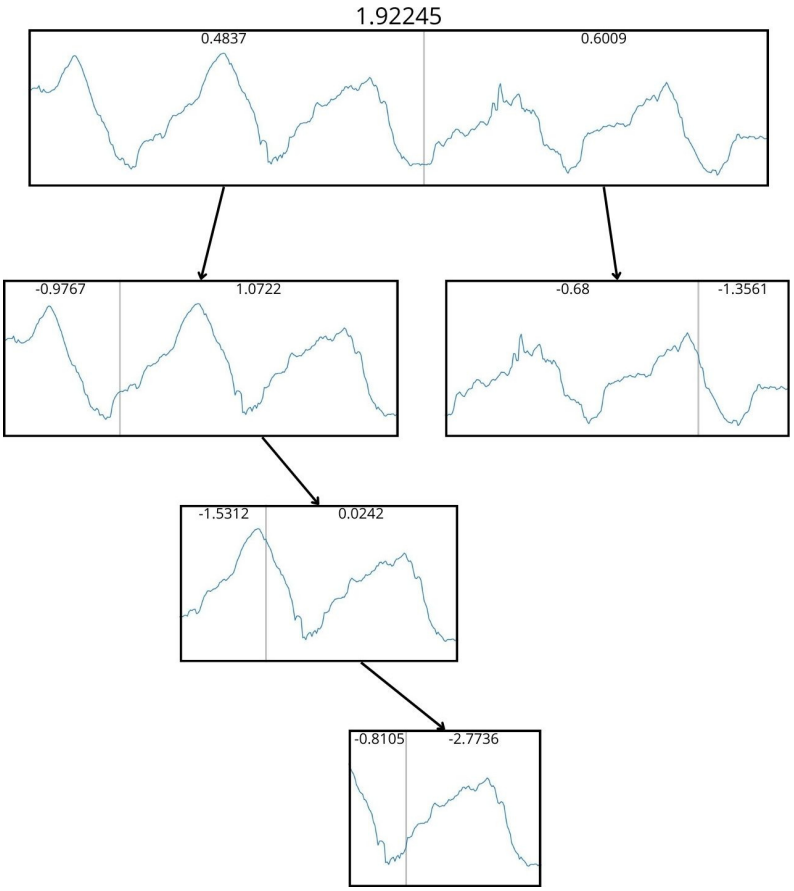


(b) Time series measuring air humidity over five days. Grey lines represent segmentation points created by XTSTree.

Figure 36 – Two time series with segments created using XTSTree.



(a) XTSTree structure created for series 1. Numbers above series and segments represent the stopping value given to that segment.



(b) XTSTree structure created for series 2. Numbers above series and segments represent the stopping value given to that segment.

Figure 37 – Two structures created by segmenting a time series.

XTSTree can guide our analysis to individual segments that have noteworthy behaviour through its structure and segmentation positions. We will first extract insights into the data by looking at the tree's structure and afterwards will analyse the series based on the information XTSTree has given us.

In Figure 37a we observe that the tree's structure is very imbalanced towards the left. This suggests a more complex behaviour on that side of the series, but knowing that the first cut was made near the end of the series (the last day of the series), this is expected. XTSTree creates cuts at positions where changes are most impactful, therefore a cut at the end of the series indicates that, if a cut had been created earlier, another one would be found near the end. The cut shown at Figure 38 happens at the end of the series, which means that either there is a drastic change near that region, or the segment to the left of the cut has some sort of change similar to what happens near the cut position, and the PH had to be tuned to ignore the first change and signal the second, otherwise it would have found two changes. Therefore, the stretch of data around this cut should be looked at. We also note that the last stretch of the series has a stopping value close to 0, which means its behaviour is not as clearly defined as other segments.

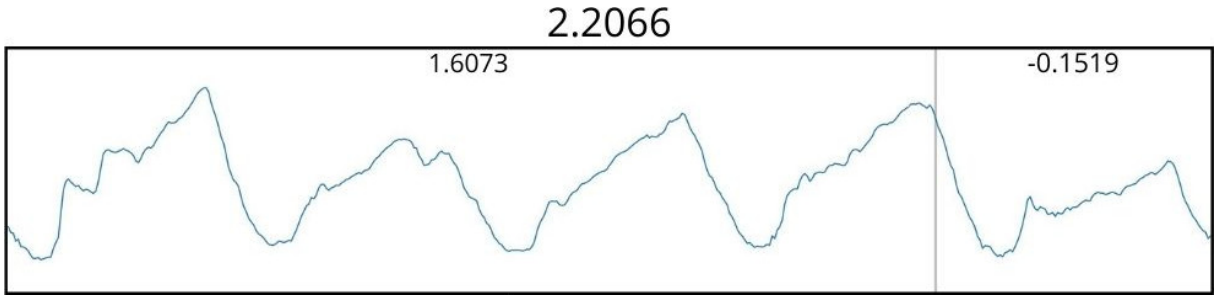


Figure 38 – Segmentation at depth 1 of the first series.

The next cut depicted by Figure 39 was created at the start of the series. Due to how PH works, we can deduce that there is a behaviour to the left of the cut that is unique when compared to the remaining subseries, meaning this segment is also of interest. This is reinforced by its stopping value being very negative. The cut created at Figure 40 lies near the middle of the series. PH finds the optimal threshold that is just enough to find a single segment in the series. If the behaviours present are sufficiently homogeneous, the cut position will happen after about half of the number of behaviours have occurred, which is usually near the middle of the series. Knowing this, we can deduce that the series cut at Figure 40 has homogeneous behaviours. XTSTree then cuts both segments, with the right segment shown in Figure 42 being cut near the middle, following the same logic as before. The final segment shown in Figure 41 is cut near the beginning of the series, resulting in a series with a stopping value greater than 0, but with a series' length too small to continue cutting. Both of these characteristics indicate that a noticeable behaviour is present at the left of the cut.

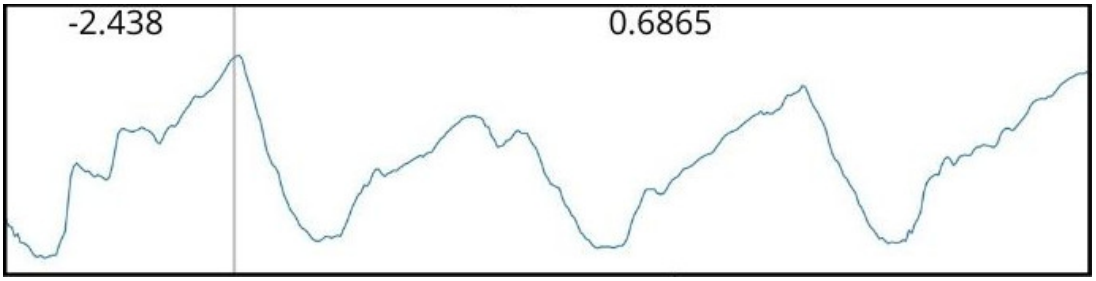


Figure 39 – Segmentation at depth 2 of the first series.

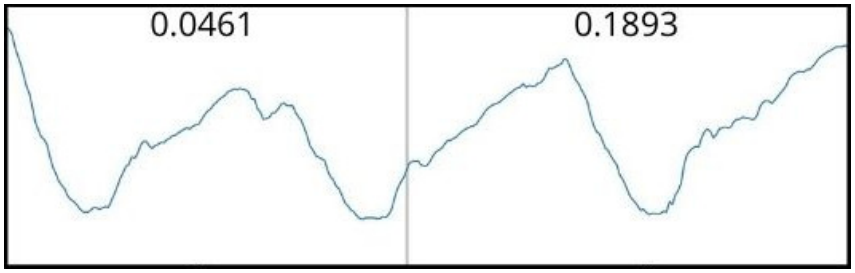


Figure 40 – Segmentation at depth 3 of the first series.

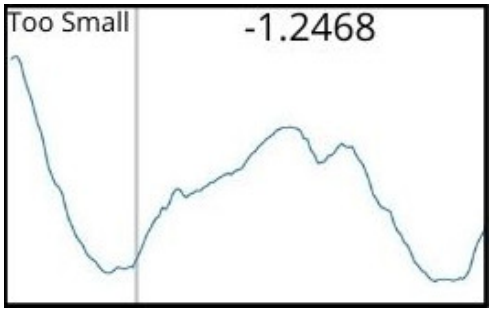


Figure 41 – Segmentation at depth 4 left side of the first series.

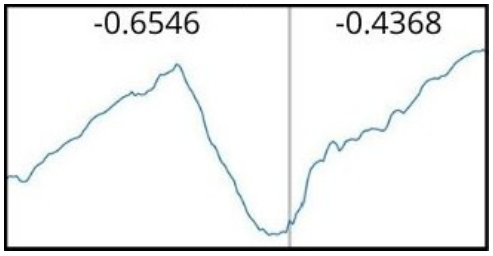


Figure 42 – Segmentation at depth 4 right side of the first series.

By analysing the tree's structure, we can conclude that readings made between 06/23, 12:15 and 06/24, 16:45 should be closely analysed, as well as readings made close to 06/27, 09:00. We can also state that the segments shown in Figure 40 has a somewhat homogeneous behaviour, and that the segment created after 06/27 may still have some interesting behaviour. Observing these periods in Figure 36a, we notice that the segment before 06/24 has the greatest increase in humidity in the smallest period. Moreover,

the segment between 06/24, 07:30 and 06/24, 16:45 has a noteworthy drop in humidity. Between 06/24, 07:30 and 06/27, 09:00, the series has a relatively stable behaviour, with the previously mentioned drop being the only outlier. The last stretch of the series appears to present two different behaviours, starting with a drop in humidity and then becoming somewhat stable. Finally, the cause behind the first segment created is likely the repeated steep drops in humidity, especially the one between 07:30 and 16:45 on 06/24.

Figure 37b also presents a left-leaning tree, however, as seen in Figure 43 the first cut was made near the middle of the series, indicating that this five-day period has a more erratic behaviour during the first half. The cut made at the segment shown in Figure 45 happens near the end of the subseries, which indicates that there is a noteworthy behaviour nearby. The cut in Figure 44 is near the beginning of the series, suggesting an analysis is needed. The same is true for the segments shown in Figure 46 and Figure 47. We can conclude that along Figure 35b, stretches between 08/08, 00:00 and 08/09, 18:00 contain interesting behaviour, as well as the stretch directly before 08/12, 9:15.

Focusing on the aforementioned areas in Figure 36b, we notice that the segment in Figure 43 has a great increase in humidity over a small period. The segment in Figure 44 has the greatest increase in humidity of the series, and the Figure 46 also has a big increase, although generally smaller, it is the greatest increase in humidity between readings. The cut position shown in Figure 45 happens during a drop in humidity, and by looking at the left segment created, we can conclude that its cause is the fast and drastic drops in humidity, as opposed to the longer periods of increasing humidity.

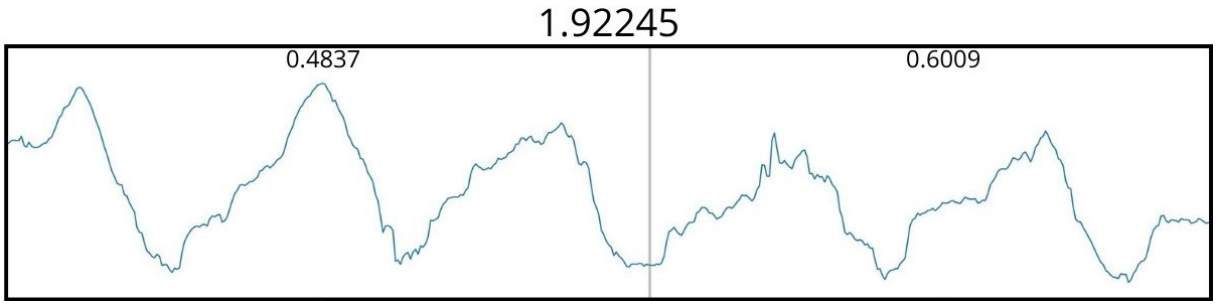


Figure 43 – Segmentation at depth 1 of the second series.

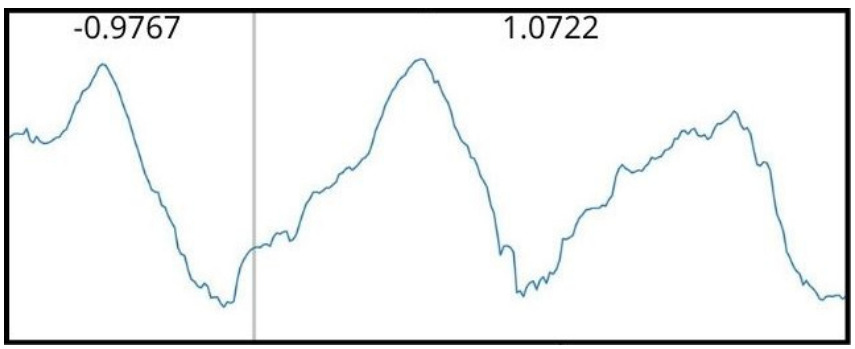


Figure 44 – Segmentation at depth 2 left side of the second series.

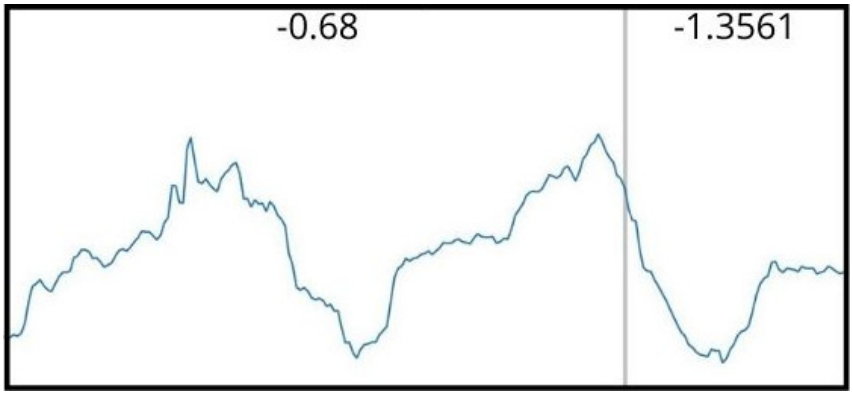


Figure 45 – Segmentation at depth 2 right side of the second series.

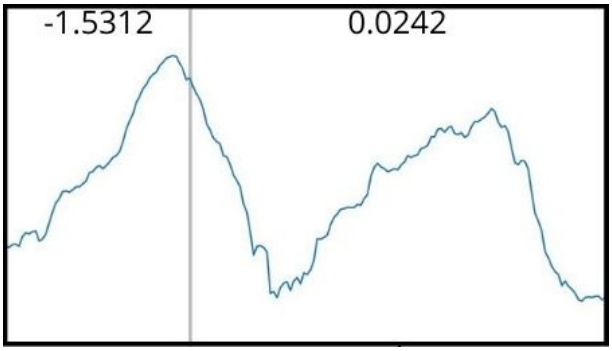


Figure 46 – Segmentation at depth 3 of the second series.

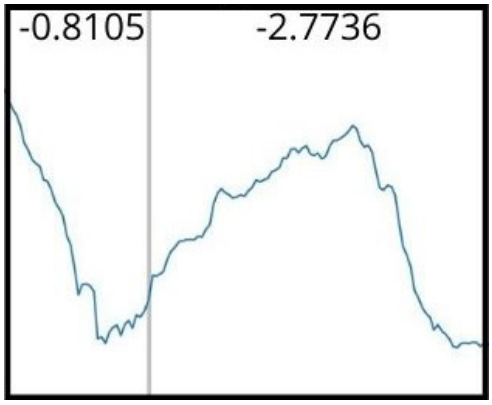


Figure 47 – Segmentation at depth 4 of the second series.

After applying XTSTree to these series, one could also use the segments created to model different, more accurate models that can more easily address each increasing or decreasing trend, as opposed to one model that leverages the complete series.

5.4 Takeaways

The result section has three main goals: validate XTSTree as a method that performs better than naive solutions while providing explainability, analyse the XTSTree's

performance compared to already consolidated top-down segmentation methods, and present an example of which valuable information XTSTree provides when applied to a real time series.

When compared with a fixed depth limit, the ADF stopping condition provides a better performance at best, and an equal performance at worst, meaning it is more scalable and requires little to no user input. The Page-Hinkley segmentation method on average is equivalent to a random segmentation method in terms of prediction error. However, the PH segmentation provides the greatest reduction in formula complexity. The PH segmentation also presents a greater mean approach towards the ADF stop condition at each segmentation than the random method. All in all, the PH and ADF methods both prove to be equal or better than naive methods.

XTSTree performs either better or similarly to other top-down segmentation algorithms, having at worst a greater error at the expense of creating fewer segments. This means that XTSTree maintains an equal performance relating to the base top-down while providing data insight and explainability. The data insight provided comes from the tree structure and segmentation method, as shown in section 5.2.

6 CONCLUSION

As the amount of time series data collected increases, so does the amount of concept drift present in these data, which is reflected in more model retraining and adaptation. Several algorithms are created specifically to identify these different behaviours and avoid a cycle of model retraining. One type of algorithm capable of identifying these changes in behaviour is time series segmentation algorithms.

Although segmentation algorithms successfully divide time series' into segments with different behaviours, they mostly use tools such as linear regression that may not be ideal to find the approximate point where the behaviour change happens. Furthermore, they seldom take into account the explainability of the cuts.

In this thesis, we proposed an algorithm for top-down segmentation that leverages explainability in the form of the XTSTree. XTSTree combines a segmentation method based on change detectors with a stopping criterion that uses a stationarity test to cut a time series into several sub-series with similar behaviour. Employing a change detector and a stationarity test allows XTSTree to approach the time series segmentation problem while leveraging the explainability of the cuts.

We made a study on related works where several studies addressing time series segmentation as a simplification process, a data insight tool and an explainability tool were described. These were classified and compared against XTSTree to highlight the differences between our method and other studies in the state of the art.

We implemented two other segmentation methods and one other stop criterion for control purposes. Performance tests using SR were used on several time series, and XTSTree managed to improve formula accuracy by a considerable amount. Among the several combinations of segmentation methods and stopping criteria, the ADF stopping criterion proved to be the most adaptable to the series' length. The Page-Hinkley segmentation method also performed better than other methods, having a statistically lower error and a more consistent number of cuts and mean leaf error while achieving a level of explainability through the hierarchy of cuts.

Two variations of the classic Top-down segmentation algorithm were implemented, one using a lag window and the other using the series' index as input. These were compared against XTSTree, with experiment results showing that the index-based top-down creates way more cuts than other methods, while the other two methods presented similar results. XTSTree stands out as the only one capable of providing insight into a time series, as well as explainability of its cuts. This is shown in a case study where it was used to guide the analyses of two weather-related time series, helping to find prominent events.

In future work, XTSTree features that provide explainability or data insight can be explicitly presented through natural language. It is also possible to implement a post-execution optimization to fine-tune the segmentation positions. Finally, there are also tree optimisations that can be done through the tree structure used to reduce the number of cuts and cluster sub-series created after segmentation.

BIBLIOGRAPHY

[1] CRANMER, M. *Interpretable Machine Learning for Science with PySR & SymbolicRegression.jl*. 2023. Disponível em: <https://github.com/MilesCranmer/pysr_paper>.

[2] SUSHMITHA, T. V. et al. Vehicle trajectory prediction using non-linear input-output time series neural network. In: *2019 International Conference on Power Electronics Applications and Technology in Present Energy Scenario (PETPES)*. [S.l.: s.n.], 2019. p. 1–5.

[3] CHIANG, P.-H.; DEY, S. Personalized effect of health behavior on blood pressure: Machine learning based prediction and recommendation. In: *2018 IEEE 20th International Conference on e-Health Networking, Applications and Services (Healthcom)*. [S.l.: s.n.], 2018. p. 1–6.

[4] NI, P.; ZHANG, C.; JI, Y. A hybrid method for short-term sensor data forecasting in internet of things. In: *2014 11th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*. [S.l.: s.n.], 2014. p. 369–373.

[5] DOKI, K. et al. Estimation of next behavior and its timing based on human behavior model with time series signal. In: *Computational Intelligence in Control and Automation (CICA)*. [S.l.: s.n.], 2011. p. 102–107.

[6] ANDERSON, R. et al. Recurring concept meta-learning for evolving data streams. *Expert Systems with Applications*, v. 138, p. 112832, 2019. ISSN 0957-4174. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0957417419305342>.

[7] LOVRIC, M.; MILANOVIC, M.; STAMENKOVIC, M. Algorithmic methods for segmentation of time series: An overview. *Journal of Contemporary Economic and Business Issues (JCEBI)*, v. 1, p. 31–53, 01 2014.

[8] WANG, C.; WANG, X. Supporting content-based searches on time series via approximation. *Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM*, p. 69 – 81, 02 2000.

[9] HUNTER, J.; MCINTOSH, N. Knowledge-based event detection in complex time series data. *Artificial Intelligence in Medicine. Joint European Conference on Artificial Intelligence in Medicine and Medical Decision Making, AIMDM'99, Aalborg, Denmark, June 1999, Proceedings*, v. 1620, p. 271–280, 06 1999.

[10] MARTí, L. et al. Yasa : Yet another time series segmentation algorithm for anomaly detection in big data problems. *HAIS- 9th International Conference on Hybrid Artificial Intelligence Systems 2014*, p. 697–708, 06 2014.

[11] KEOGH, E.; CHU, S.; PAZZANI, M. Segmenting time series: A survey and novel approach. *Data Mining in Time Series Databases*, v. 57, 03 2003.

[12] SEBASTIAO, R.; FERNANDES, J. M. Supporting the page-hinkley test with empirical mode decomposition for change detection. In: *International Syposium on Methodologies for Intelligent Systems*. [S.l.: s.n.], 2017. p. 492—-498.

[13] DICKEY, D.; FULLER, W. Distribution of the estimators for autoregressive time series with a unit root. *JASA. Journal of the American Statistical Association*, v. 74, 06 1979.

[14] SILVESTRE, J. Carrion-i; SANSO, A. A guide to the computation of stationarity tests. *Empirical Economics*, v. 31, p. 433–448, 02 2006.

[15] HYNDMAN, R.; ATHANASOPOULOS, G. *Forecasting: principles and practice*. OTexts, 2018. ISBN 9780987507112. Disponível em: <https://books.google.com.br/books?id=_bBhDwAAQBAJ>.

[16] LOPEZ, J. The power of the adf test. *Economics Letters*, v. 57, n. 1, p. 5–10, 1997. ISSN 0165-1765. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0165176597818721>.

[17] MACKINNON, J. Critical values for cointegration tests. *Long-Run Economic Relationships*, 02 1990.

[18] GAMA, J. a. et al. A survey on concept drift adaptation. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 46, n. 4, mar 2014. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/2523813>.

[19] KRAWCZYK, B. et al. Ensemble learning for data stream analysis: A survey. *Inf. Fusion*, v. 37, p. 132–156, 2017. Disponível em: <https://api.semanticscholar.org/CorpusID:1372281>.

[20] GAMA, J. a. et al. A survey on concept drift adaptation. *ACM Comput. Surv.*, Association for Computing Machinery, New York, NY, USA, v. 46, n. 4, mar 2014. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/2523813>.

[21] HALLAC, D. et al. *Toeplitz Inverse Covariance-Based Clustering of Multivariate Time Series Data*. 2018.

[22] CHEN, Y.; HUANG, S. *TSEXPLAIN: Explaining Aggregated Time Series by Surfacing Evolving Contributors*. 2022.

[23] KIM, B.; KHANNA, R.; KOYEJO, O. O. Examples are not enough, learn to criticize! criticism for interpretability. In: LEE, D. et al. (Ed.). *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2016. v. 29. Disponível em: <https://proceedings.neurips.cc/paper_files/paper/2016/file/5680522b8e2bb01943234bce7bf84534-Paper.pdf>.

[24] MILLER, T. *Explanation in Artificial Intelligence: Insights from the Social Sciences*. 2018.

[25] NAUTA, M. et al. From anecdotal evidence to quantitative evaluation methods: A systematic review on evaluating explainable ai. *arXiv preprint arXiv:2201.08164*, 2022.

[26] MOLNAR, C. *Interpretable Machine Learning*: A guide for making black box models explainable. 2. ed. [s.n.], 2022. Disponível em: <https://christophm.github.io/interpretable-ml-book>.

[27] BURKART, N.; HUBER, M. F. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research*, AI Access Foundation, v. 70, p. 245–317, jan. 2021. ISSN 1076-9757. Disponível em: <http://dx.doi.org/10.1613/jair.1.12228>.

[28] MOLLEL, R. S.; STANKOVIC, L.; STANKOVIC, V. Using explainability tools to inform nilm algorithm performance: a decision tree approach. In: *Proceedings of the 9th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation*. New York, NY, USA: Association for Computing Machinery, 2022. (BuildSys '22), p. 368–372. ISBN 9781450398909. Disponível em: <https://doi.org/10.1145/3563357.3566148>.

[29] LANGDON, W. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!* Springer US, 1998. (Genetic Programming). ISBN 9780792381358. Disponível em: <https://books.google.com.br/books?id=MyNvEee9tDgC>.

[30] KOZA, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Bradford, 1992. (A Bradford book). ISBN 9780262111706. Disponível em: <https://books.google.com.br/books?id=Bhtxo60BV0EC>.

[31] CAVA, W. L. et al. *Contemporary Symbolic Regression Methods and their Relative Performance*. 2021.

[32] VIRGOLIN, M. et al. Model learning with personalized interpretability estimation (ml-pie). In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. New York, NY, USA: Association for Computing Machinery, 2021. (GECCO '21), p. 1355–1364. ISBN 9781450383516. Disponível em: <https://doi.org/10.1145/3449726.3463166>.

[33] TONEKABONI, S. et al. *What went wrong and when? Instance-wise Feature Importance for Time-series Models*. 2020.

[34] RIBEIRO, M. T.; SINGH, S.; GUESTRIN, C. *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. 2016.

[35] ALBINI, E. et al. Relation-based counterfactual explanations for bayesian network classifiers. In: BESSIERE, C. (Ed.). *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*. International Joint Conferences on Artificial Intelligence Organization, 2020. p. 451–457. Main track. Disponível em: <https://doi.org/10.24963/ijcai.2020/63>.

[36] POCHé, A.; HERVIER, L.; BAKKAY, M.-C. *Natural Example-Based Explainability: a Survey*. 2023.

[37] DOSHI-VELEZ, F.; KIM, B. *Towards A Rigorous Science of Interpretable Machine Learning*. 2017.

[38] YASODHARA, A. et al. On the trustworthiness of tree ensemble explainability methods. In: _____. *Machine Learning and Knowledge Extraction*. Springer International Publishing, 2021. p. 293–308. ISBN 9783030840600. Disponível em: <http://dx.doi.org/10.1007/978-3-030-84060-0_19>.

[39] LI, G. et al. A new piecewise linear representation method based on the r-squared statistic. In: *2021 3rd International Conference on Machine Learning, Big Data and Business Intelligence (MLBDBI)*. [S.l.: s.n.], 2021. p. 515–519.

[40] WEE, C. K.; NAYAK, R. Alternate approach to time series reduction. In: *2018 International Conference on Soft-computing and Network Security (ICSNS)*. [S.l.: s.n.], 2018. p. 1–4.

[41] FILLON, C.; BARTOLI, A. Symbolic regression of discontinuous and multivariate functions by hyper-volume error separation (hves). In: *2007 IEEE Congress on Evolutionary Computation*. [S.l.: s.n.], 2007. p. 23–30.

[42] MIN, H.; LEE, J.-G. Temporal convolutional network-based time-series segmentation. In: *2023 IEEE International Conference on Big Data and Smart Computing (BigComp)*. [S.l.: s.n.], 2023. p. 269–276.

[43] MATSUBARA, Y.; SAKURAI, Y.; FALOUTSOS, C. Autoplait: Automatic mining of co-evolving time sequences. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 06 2014.

[44] GHARGHABI, S. et al. Matrix profile viii: Domain agnostic online semantic segmentation at superhuman performance levels. In: . [S.l.: s.n.], 2017. p. 117–126.

[45] YEH, C.-C. M. et al. Matrix profile i: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. [S.l.: s.n.], 2016. p. 1317–1322.

[46] SIVILL, T.; FLACH, P. Limesegment: Meaningful, realistic time series explanations. In: CAMPS-VALLS, G.; RUIZ, F. J. R.; VALERA, I. (Ed.). *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*. PMLR, 2022. (Proceedings of Machine Learning Research, v. 151), p. 3418–3433. Disponível em: <https://proceedings.mlr.press/v151/sivill22a.html>.

[47] MURALIDHAR, N. et al. Cut-n-reveal: Time-series segmentations with explanations. *ACM Transactions on Intelligent Systems and Technology*, v. 11, 05 2020.

[48] ROJAT, T. et al. *Explainable Artificial Intelligence (XAI) on TimeSeries Data: A Survey*. 2021.

[49] GILPIN, L. H. et al. *Explaining Explanations: An Overview of Interpretability of Machine Learning*. 2019.

[50] BEZANSON, J. et al. Julia: A fresh approach to numerical computing. *SIAM Review*, v. 59, n. 1, p. 65–98, sep 2017.

PAPERS SUBMITTED BY THE AUTHOR

Main Submission.

1. Vitor de Castro Silva, Bruno Bogaz Zarpelão, Eric Medvet e Sylvio Barbon, **Explainable Time Series Tree: An explainable top-down time series segmentation framework**, IEEE Access, October/2023, IEEE, 120845-120856, 2169-3536, 10.1109/ACCESS.2023.3326847 (Qualis CC 2017, A3).

Complementary Submissions.

1. Matheus Matiazzo, Vitor de Castro Silva, Rafael Oyamada and Daniel Kaster, **Usage of Datasets for Evaluation in Similarity Retrieval: Popular Choices and a New Similarity-based Dataset Selection Approach**, SISAP 2023: Similarity Search and Applications, October/2023, Springer, Cham, 125-132, 978-3-031-46994-7, 10.1007/978-3-031-46994-7_11.

2. Vitor de Castro Silva, Wagner Ferreira Lima, Cinthyan de Barbosa. **Léxico Rural da Serra do Cipó: um caso de indexação digital de variantes regionais do português.**, Computer on the Beach 2023, May/2023, Computer on the Beach, 296-302, 2358-0852, 10.14210/cotb.v14.p296-302 (Qualis CC 2017, B4).