Politecnico di Torino

Mathematical Data Science Engineering

A.Y. 2023/2024

OD to detect transitions between time series segments

Master's Degree Thesis

Supervisors:
Luca CAGLIERO

Author:
Ilaria ZERBINI

November 13, 2023

Summary

Time series data, prevalent across scientific studies, captures the temporal evolution of phenomena such as sports activities, weather patterns, and health conditions. The fundamental task of classifying these data hinges on identifying statistical properties that can be used to predict labels. This becomes particularly challenging with long, multi-dimensional time series, leading to the necessity of a previous step, in which the segmentation into sub-series is done. This research focuses on recognizing transitions between segments characterized by homogeneous trends, a critical aspect of segmentation. Transition points are often ambiguous, situated at the juncture of two states, and identifying them poses unique segmentation challenges. In this thesis work, we will address this task by exploring supervised and semi-supervised outlier techniques, exploring the new task at hand. Our investigation will begin by establishing the performance of supervised methods, introducing a new tree ensemble method, coupled with a novel sampling strategy. Then, for the transition recognition, we will leverage Time2State, an unsupervised state-of-the-art contrastive learning architecture for time series segmentation. This will be expanded by integrating an unsupervised outlier detection and a separation score-based sorting criterion, addressing real-world semi-supervised scenarios. Additionally, we will explore data transformation using Shapelets to represent time series data. Thorough testing on real and synthetic datasets demonstrates that Time2State alone struggles with precise transition recognition. However, the supervised pipeline performs solidly, and the semi-supervised pipelines outperform the sole Time2State, providing valuable domain insights.

Acknowledgements

Questo elaborato segna l'epilogo di un percorso lungo cinque anni, un percorso che ha portato con sè tante emozioni di ogni tipo.
Non è stato un viaggio semplice. Ho intrapreso questa sfida quasi a voler dimostrare a me stessa che forse, con impegno e accanimento, posso arrivare dove vorrei. Probabilmente è proprio questa testardaggine che mi ha portato dove sono ora.
Non è iniziato per una passione particolare, ma seguendo passo passo l'esempio di mio zio Mario sin dalla scelta del liceo classico e dopo del Politecnico. Il suo ricordo mi ha sempre accompagnato, ricordandomi che tutto il sudore versato, presto o tardi, avrebbe dato i suoi frutti; e come sempre, aveva ragione lui.
Sarebbe difficile trovare un qualsiasi membro della mia famiglia da non ringraziare. I miei genitori che hanno sempre avuto come priorità il mio benessere, che hanno sempre creduto ciecamente in me e mai, mai hanno dubitato che potessi riuscire. Mio fratello, forse il più orgoglioso di tutti, che mi ha sempre portato su un piedistallo davanti a chiunque tranne che a me, per essere sicuro di infastidirmi, come piace fare a lui. Il mio ragazzo Sato, che mi ha accompagnato per tutto questo viaggio, sostenendomi nei momenti più difficili, quando lui era il solo a poter comprendere la frustrazione e le gioie che provavo. Non per ultimi i miei zii e le nonne che, ognuno a modo suo, mi hanno sempre ricordato quanto valessi, di essere fiera e camminare a testa alta.
Non so se a posteriori farei le stesse scelte, non concludo questa avventura senza rimpianti, ma sicuramente sono felice di poter dire a tutti coloro che in me ci hanno creduto dall'inizio: "Ce l'ho fatta".
Quasi dimenticavo, i ringraziamenti più sentiti vanno a Melo, il mio cagnone, che è sempre stato lì, vicino a me, nelle notti di studio, nei pianti e nelle gioie più grandi.

Table of Contents

List of Figures

Chapter 1

Introduction

In all types of science-based studies, data is a common element. Everyone has used a histogram or a scatter plot at least once. With the ever-increasing trend of accumulating large amounts of data, time series are more present than ever in our daily lives.
Most things, such as sports, weather, and even health conditions, present a strong link with the flowing of time, permitting us to record the change in their status in the form of time series [27]. This data mode is the most widely used, since it permits to describe in a clear and comprehensible way, how a certain observed phenomenon is evolving during the passage of time. Therefore, finding new ways to approach and work with this kind of data representation, has become a necessity. The most common task in time series analysis is classification [9], usually done by identifying statistical and behavioral properties in the data. This allows to predict the label of the specific series under study.
There are numerous tools [6] available for time series analysis, as this problem has been extensively explored in recent years. Particularly, it is usual to end up working with long multi-dimensional series, making this task even harder. This comes from the necessity to first preprocess them, segmenting the series into shorter sub-sequences, and then solving the classification task on those in an easier and more significant way (Figure 1.1 shows an exemplary situation of the sort).

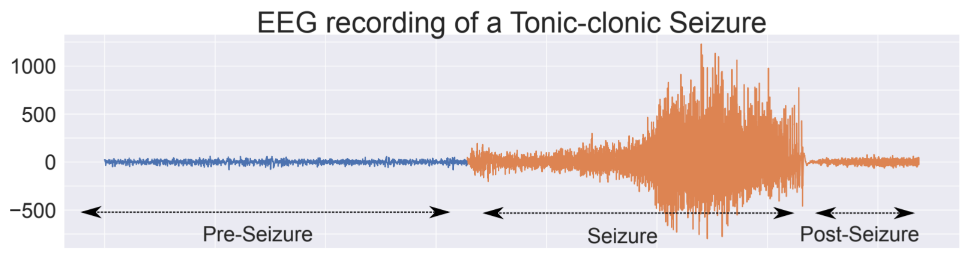


Figure 1.1: The image shows an ECG recording a case of Tonic-clonic Seizure [45], during which it is possible to identify three sub series linked to three more specific activities.

Let's consider a long time series that records the movements, such as the 3D acceleration, of a person training in the gym. His activity can be divided into sub-series, each labeled differently, such as "jumping," "doing plank," "running," and so on. In this scenario, we are placed before a choice in order to accomplish the classification task: it is possible to either try to segment the series, aiming to distinguish between these macro-activities, or the focus can be on recognizing the moments in which the subject changes his state, without being concerned about the specific activity he was performing.
The most delicate part in this analysis is surely the identification of the transitions between states. These instants are cloudy, since they are on the verge of abandoning the previous condition and on the edge of entering another one. Therefore, their behavior and statistical properties have similarities to both classes.
No matter which we choose to address, we find ourselves before a complex task, which becomes even harder when the data at our disposition isn't periodic but random or without a recognizable logic behind its change of states.
However, between the two, the segmentation task is surely more known, and some literature and studies regarding it are already available [5]. Mainly, the methods used in this field can be categorized as traditional or neural based ones.
The focus of this work is the recognition of transition between segments characterized by homogeneous trends, aiming to employ supervised and semi-supervised outliers techniques in order to recognize such transitions and, as a possible future work, improving the results of the segmentation task.
The first step of our work consisted in exploring supervised techniques, which led us to implement a new method based on trees ensemble, combined with a novel sampling strategy.
Then, our analysis continues with a deep dive into the potential of Time2State [3], an unsupervised state-of-the-art method used in the context of time series segmentation. Thanks to the logical link between the two tasks, the algorithm can be easily adjusted to solve both; after this first study, we aimed to find ways to

enhance its performances, creating a more complex pipeline.
Hence, we tried to augment the Time2State workflow with unsupervised Outlier Detection [47] algorithms and a sorting criterion based on separation score between different state clusters.
The background for the latter is conceived as a semi-supervised scenario, in which we find the predicted labels using Time2State on the series up to a certain point in time, after which we are completely blind, losing the possibility to train our model.
We have also explored the possibility to transform the data using the Shapelets [43], rewriting a series as the distance between its most characteristic shapes.
We have tested our pipelines on ActRecTut [30], a dataset about human activities, and one hundred synthetic datasets.
The analysis conducted demonstrates that Time2State does not perform well when it comes to recognizing transitions with a certain level of precision, proving the necessity of finding new ways to tackle the problem at hand.
Moreover, the supervised pipeline obtained good and solid results, despite its intrinsic simple architecture.
Finally, for the semi-supervised pipelines, the obtained results easily outclass those of Time2State in the case of ActRecTut, and are better or equal to the synthetic datasets, while also providing important domain information.

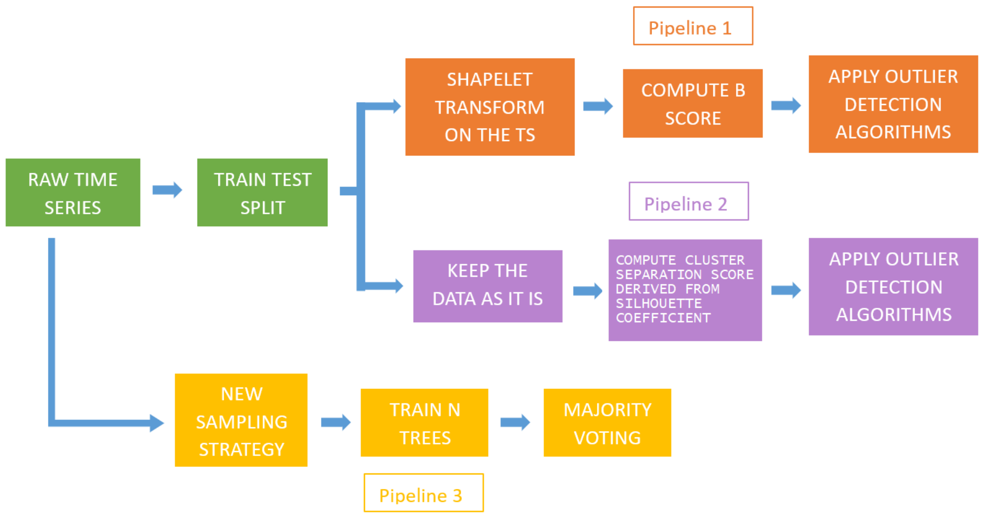

Pipeline 1
SHAPELET TRANSFORM ON THE TS
COMPUTE B SCORE
APPLY OUTLIER DETECTION ALGORITHMS
RAW TIME SERIES
TRAIN TEST SPLIT
Pipeline 2
KEEP THE DATA AS IT IS
COMPUTE CLUSTER SEPARATION SCORE DERIVED FROM SILHOUETTE COEFFICIENT
APPLY OUTLIER DETECTION ALGORITHMS
NEW SAMPLING STRATEGY
TRAIN N TREES
MAJORITY VOTING
Pipeline 3

Figure 1.2: High level summary of the new pipelines proposed in this work

Chapter 2

Background Knowledge

2.1 What is a Time series

When dealing with a measurable phenomenon, in order to be able to capture its behaviour, we usually register, choosing the desired temporal granularity, its changes it time. We could be interested not only in how one of its characteristics develops (univariate time series), but also how multiples ones do (multivariate time series). In both cases chosen t=t_0 as the starting time for our recordings, we define the time series (TS) S as the sequence of observed changes in the phenomenon we are studying. Each of these observations are indicated as x_{ij}, where i is the index linked of the feature of the phenomenon we are measuring, and j is its temporal index.
Therefore we will have also t=T as the last time index of the recordings, from which we compute as T-t_0 the length of the series.
As said earlier in the Introduction [1], the regular task linked to this data format is classification; however, in our case we will consider TS longer than usual, such that they can be further divided into shorter sub-series (for reference see Figure (2.1)). To these sub-sequences, also called states, we associate a simple event, occurrence or activity which will represent our new aim in the intra-series classification.
Hence, given a TS S = $\{x_i\}_{i=0}^{T}$, with $x_i \in \mathbf{R^{d \geq 1}}$, the sub-sequence $x_{i:j}$, $t_0 \leq i < j \leq T$, spans a certain state $A \in \mathbf{C}$, where $\mathbf{C}$ is the set of all possible states for S, if:

- the class A is recognized by distinguishable and repetitive patterns;
- all the samples $x_{i:j}$ contains belong to the same class A;

Therefore, to each sample x_i of S, we associate a class s_i, calling s = $\{s_i | i \in \mathbf{N}^+, i \leq T, s_i \in \mathbf{C}\}$ the list of the states spanned by the series.

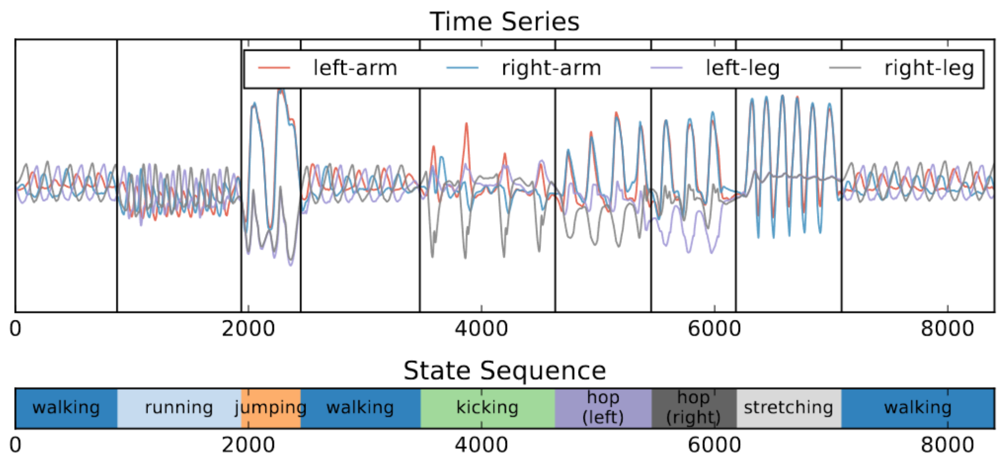


Figure 2.1: Example of time longer multivariate time series, split into its subsequences. The various class clearly show different characteristics [3].

2.2 Machine Learning

In recent years, we have witnessed the exponential growth of machine learning [39], and its ever-increasing integration into our daily lives. The potential applications of this technology in the field of engineering are virtually limitless, underscoring its undeniable utility.
To be more precise, machine learning is a sub-field of Artificial Intelligence, another prominent player in the modern era, which equips computers with the remarkable capability to learn and adapt. The learning process, in particular, consists of two distinct phases:

- We input data into the computer, enabling it to "learn" from it;
- With the knowledge acquired, and without the need for further specific programming, the computer autonomously makes decisions, such as classifying or predicting new unknown data;

This operation is fundamentally rooted in how humans themselves approach the learning process: by identifying patterns, statistical properties, and adapting our perspectives to new information.
Therefore, machine learning algorithms function as "data-driven" learners, specifically designed to process extensive datasets, uncover underlying structures, and employ this knowledge to make decisions when confronted with new, unseen data.
Machine learning has given rise to various specialized branches, each tailored for

specific tasks. To gain a deeper understanding of the subjects covered in this work, we will now briefly introduce three different forms of the learning background: Supervised Learning, Unsupervised Learning, and Semi-supervised Learning.

2.2.1 Supervised Learning

Supervised Learning [40] is the first and most common approach used in machine learning.
In this context, we give to the algorithm a training dataset X of length N, which contains various samples x_i, with $0 \leq i \leq N$, and the label or groundtruth y_i associated to every observation. The "supervised" adjective comes from this last specific, as the supervisor to the algorithm gives it the "solution" for a part for the data, so that the model can tune itself during the learning process.
With the provided input, the algorithm starts its training phase, which consists in finding a function f(X) that maps the training data X to its predicted labels Y, trying to grasp their underlying connection, committing the least amount of errors based on the known information.
This is an iterative process, such that the mapping function is adjusted during each following run, led by the indications of a chosen loss function [20]. As the name would suggest, the scope of this function is "punish" the algorithm whenever it wrongly labels a sample, encouraging it to go towards the right solution.
An interesting feature of this approach, is that it can be adapted to both classification and regression tasks; moreover there is a limitless list for loss functions, which can also be made ad-hoc for the type and distribution of the data or the task at hand.
Some notable representatives of this class are: Artificial Neural Network [10], Logistic Regression [19], Decision Trees [14] and Support Vector Machines [26].

2.2.2 Unsupervised Learning

Unsupervised learning [28] can be viewed as the polar opposite of the aforementioned case, as its fundamental premise revolves around being "unsupervised".
Specifically, in this context, we provide the algorithm solely with the data X, omitting the associated ground truth. Consequently, it does not depend on knowledge derived from prior analysis conducted on labeled data.
Given this background, in the best-case scenario, for the same data X and task, these algorithms are expected to perform less effectively or equally as compared to supervised learning methods. On the flip side, while unsupervised learning might not match the performance of supervised methods in terms of predictive accuracy, it offers the advantage of requiring minimal expert intervention, making it an essential tool in various real-world applications.

Usually we apply this algorithms when doing:

- ASSOCIATION RULES [11]: We explore the dependencies between data items and map their results in a way that suggests increased profitability when these items are considered together. An example of this is the Apriori Algorithm [37];
- CLUSTERING [38]: This approach attempts to organize given data points into sets where the elements within each set share specific statistical properties. Notable algorithms for clustering include DBSCAN [13], K-means [17], and Spectral Clustering [25];
- ANOMALY DETECTION: The goal here is to identify data points that deviate from the norm, meaning they are outliers in our data distribution. Algorithms like Local Outlier Factor [18] and Isolation Forest [16] are commonly employed for this purpose

Hence, unsupervised learning, in stark contrast to supervised learning, operates without the luxury of ground truth data.
It is a domain of machine learning where algorithms are designed to uncover hidden patterns and relationships within datasets, making it particularly useful in scenarios where labeled data is scarce or non-existent.

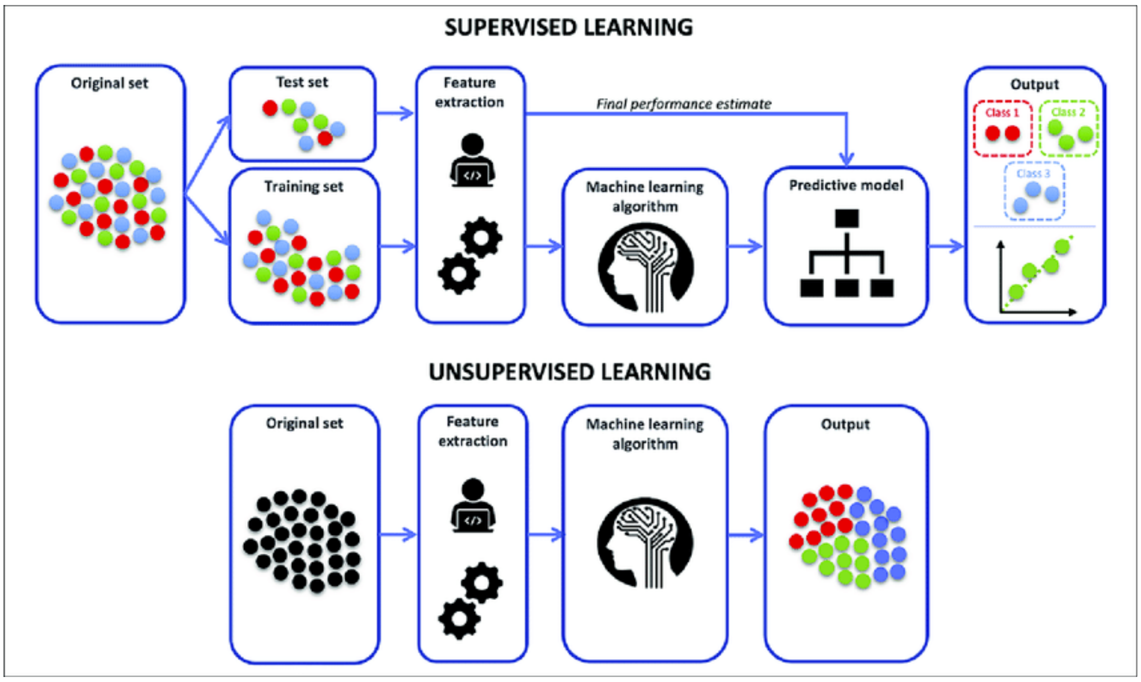


Figure 2.2: Summary scheme which shows the differences between unsupervised and supervised learning[4]

2.2.3 Semi-supervised Learning

Semi-supervised learning [29], also called weak supervision, occupies the middle ground between the two previously discussed approaches.
During the training phase, we supply the algorithm with a dataset X that is not fully labeled, aiming to extract the latent properties of the observations. By doing so, we address the challenge of obtaining or creating a sufficiently large labeled dataset, while simultaneously enhancing the performance of simpler unsupervised algorithms.
This approach is particularly valuable when we have access to only a limited number of labeled samples but have abundance of unlabeled data. One common scenario where semi-supervised learning shines is in the realm of Natural Language Processing [21].
In NLP, it is often impractical or too costly to manually label vast amounts of text data. Semi-supervised learning allows us to make the most of the labeled data we do have, along with the abundance of unlabeled text, to build effective models for various language-related tasks. This approach strikes a balance between the supervised and unsupervised methods, making it a valuable tool in scenarios where labeled data is scarce, but leveraging unlabeled data can significantly improve model performance.

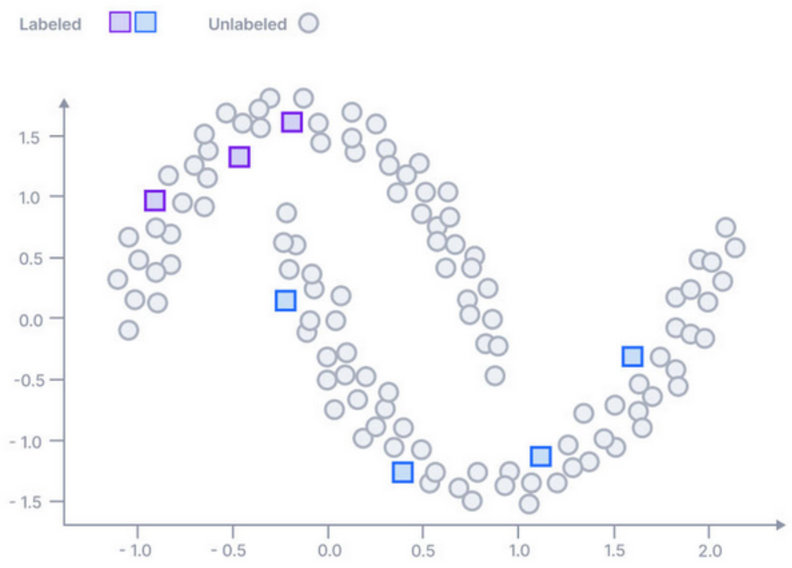


Figure 2.3: Example of semi-supervised scenario, in which we have a few labelled samples, while the majority is unlabelled [41]

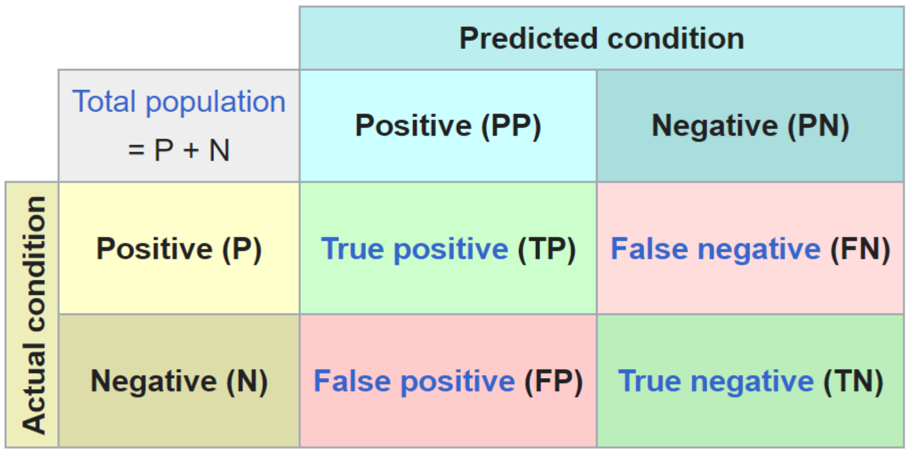
2.3 Performance Measures for Binary Classification tasks

In binary classification tasks, evaluating the performance of a machine learning model is paramount for gauging its effectiveness in making predictions. In particular, the confusion matrix[12], often referred to as Error matrix, is a vital tool in this process. It provides a structured framework to systematically analyze the model's predictions and compare them to the ground truth labels.
The confusion matrix is also the starting point used to compute and create numerous measures used in machine learning. Those measures permit us to better understand how our approach is performing, focusing on specific tasks and aspects, and permitting us to get the full picture of its flaws and strengths.
For the purpose of this work, we relied on two specific indexes: Precision and Recall. These metrics provide a nuanced perspective on a model's ability to make accurate positive predictions and capture all positive instances.
Therefore, thanks to them, data scientists can make informed decisions regarding the model's suitability for a given task and the trade-offs between minimizing false alarms and capturing all relevant positive cases.

2.3.1 Confusion Matrix

The confusion matrix [12] plays a pivotal role in evaluating binary classification models. It is composed of four primary elements: true positives, true negatives, false positives, and false negatives. Each of these elements holds specific meaning in the context of assessing model performance.
For binary classification, it consists in a 2x2 matrix that summarizes the classification results, composed as follows:

- True Positives (TP): These are the cases where the model correctly predicted a positive class when the actual class was positive.
- True Negatives (TN): These are the cases where the model correctly predicted a negative class when the actual class was negative.
- False Positives (FP): These are the cases where the model incorrectly predicted a positive class when the actual class was negative. Also known as Type I errors.
- False Negatives (FN): These are the cases where the model incorrectly predicted a negative class when the actual class was positive. Also known as Type II errors.



		Predicted condition	
	Total population = P + N	Positive (PP)	Negative (PN)
Actual condition	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Figure 2.4: Example of confusion matrix [12]

2.3.2 Precision and Recall

Precision is a metric that focuses on the accuracy of positive predictions made by a model. It quantifies the proportion of true positive predictions in comparison to all positive predictions generated by the model. Achieving a high precision score is crucial for tasks where **false positives** can have substantial consequences, such as medical diagnoses and fraud detection.
It is calculated as the ratio of true positives to the total number of positive predictions made by the model:

$$Precision = \frac{TP}{(TP + FP)}$$

A high precision score indicates that when the model predicts the positive class, it is likely to be correct, minimizing false alarms. However, precision does not account for missed positive cases, making it crucial to consider it in conjunction with recall. Meanwhile the recall, often referred to as sensitivity or true positive rate, is a metric that emphasizes a model's ability to capture all positive instances. It measures the proportion of true positive predictions in relation to all actual positive cases.
High recall is desirable in scenarios where **missing** positive cases can have critical implications, such as search and rescue operations, but it may also lead to higher false alarms if precision is low.

$$Recall = \frac{TP}{(TP + FN)}$$

For the purpose of this work, we've selected these measures as indicators of performance for our algorithms for two main reasons:

- We aim to identify the greatest number of positive samples, which, in our task, constitute the minority class. Therefore recall suits the task at hand;
- Additionally, we aim to avoid mislabeling negative samples as positives, as this could provide crucial information about other positive samples. Hence, we seek certainty in labeling samples as positives, which can be monitored by precision;

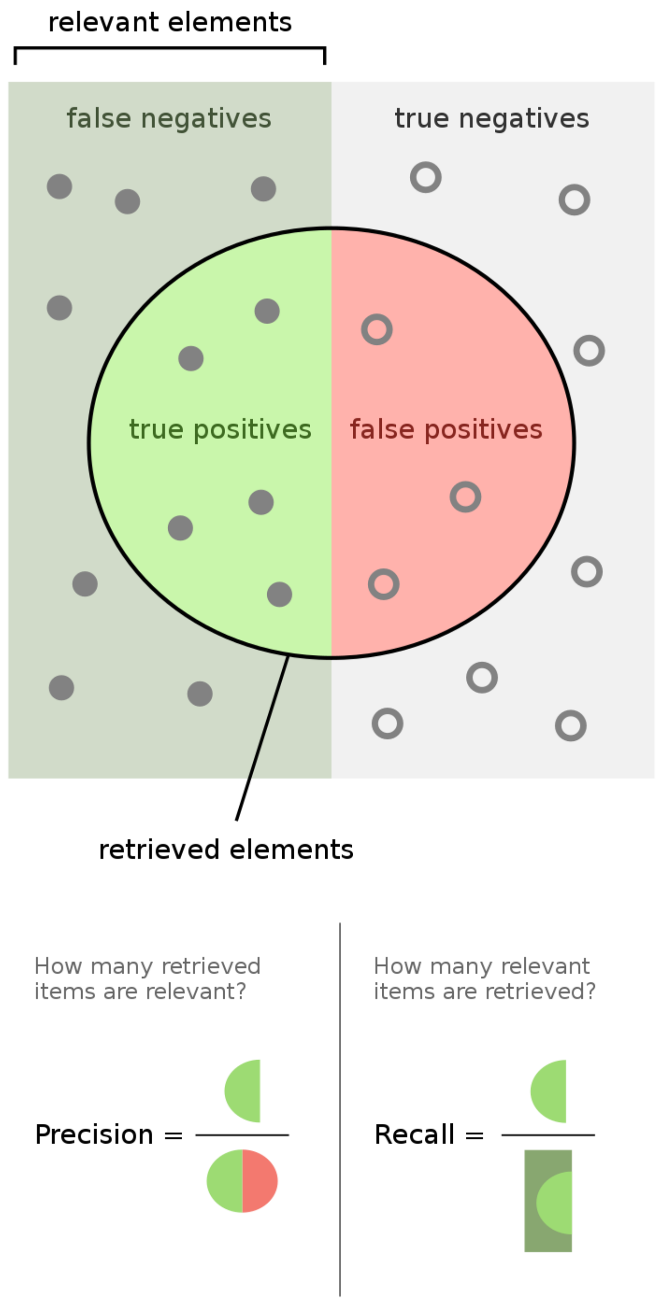


Figure 2.5: Visual example of how recall and precision are computed [22]

2.3.3 The Precision-Recall Trade off

Precision and recall are often in tension with each other, meaning that improving one can adversely affect the other.
Understanding this trade off is crucial when tuning models and making decisions in real-world applications. Achieving an optimal balance between precision and recall is context-dependent and requires a thorough understanding of the problem at hand.

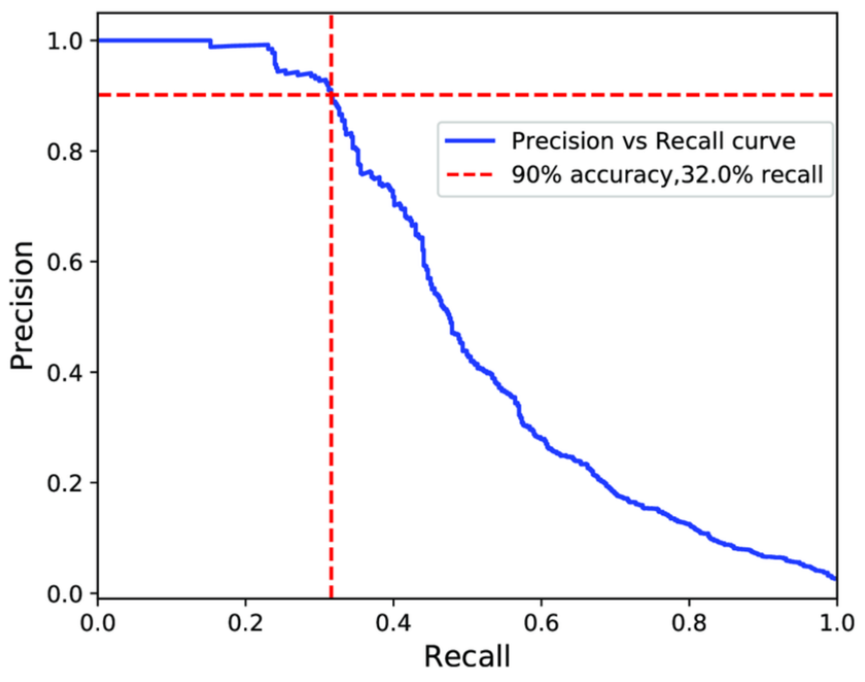


Figure 2.6: A classic precision-recall plot [1]

Chapter 3

State of the Art methods for Outlier detection and Time Series Segmentation

3.1 Outlier Detection

Outlier Detection, is a key machine learning task with numerous applications in modern days, such as rare disease detection [49], social media analysis [50] and fraud detection. Its objective is to identify samples in the data that deviate significantly from the majority of the distribution [8, 46]. A clear overview of the current state of the most promising approaches used in the field is provided in "ADBench: Anomaly Detection Benchmark", by Songqiao Han et all. [51], where the majority of the benchmarks are listed and compared in various background conditions, from supervised to unsupervised.
Depending on the level of supervision used we have the following categories of methods:

- UNSUPERVISED: These methods rely on assumptions about the distribution of anomalies, such as supposing they are located in low-density regions. The main distinction between the various approaches consists of differences in these assumptions, decisions that can lead to varying performance depending on how well these choices align with the real data. There are mainly two subcategories:

  1. Shallow methods: These methods provide better interpretability of the results requiring no particular prior information on the data. In this work, we will use three algorithms from this group: LOF, IForest, and OCSVM [31];

2. Deep Neural Network methods: These methods exploit complex architectures to autonomously discern anomalies based on inherent patterns and irregularities within the data. Their strength lies in handling vast, high-dimensional datasets efficiently, enabling the detection of anomalies that might evade traditional methods. Moreover thanks to deep learning techniques, these methods can adapt to various data structures, providing a versatile solution. However, due to the intricate structure and substantial parameter fine-tuning required, we've opted not to utilize them for our current purpose;
3. Matrix Profile Based methods: These methods represent a stand as a powerful tool in time series analysis, primarily employed for outlier detection. They are based on the concepts of motifs and discords discovery in a series. Their best feature is the swift computation of nearest neighbor distances, which permits the precise identification of anomalies nestled within time series data. In this work we have analyzed one algorithm from this category, called DAMP [44];

- SEMI-SUPERVISED: Semi-supervised methods tries to efficiently use the partially available labels, in order to also keep the ability to detect unseen types of anomalies. For example some semi-supervised models are trained only on normal samples, while are tested on anomalies that deviate from the normal representations learned in the training process;
- SUPERVISED: These methods see the problem in a binary fashion, meaning that we have something similar to a binary classification task. Thanks to the accessibility of ground truth labels, supervised classifiers may identify known anomalies at the risk of missing unknown ones. In this case, standard classification approaches are used, as Random Forest or Neural network.

3.1.1 Local Outlier Factor

The main scope of the approach, called in short LOF, is finding anomalous data points by measuring the local deviation of a given data point with respect to its neighbours. It results to be effective in identifying anomalies in high-dimensional datasets.
LOF operates on the principle of measuring the local density of data points within a given dataset. In essence, it assesses how similar a data point is to its neighbours by assigning an anomaly score to it, called Local Outlier Factor. It is local since the anomaly score depends on how isolated the sample is with respect to the surrounding neighborhood. More precisely, locality is given by k-nearest neighbors, whose distance is used to estimate the local density. By comparing it to the local

densities of its neighbors, one can identify samples that have a substantially lower density, and label those as outliers.

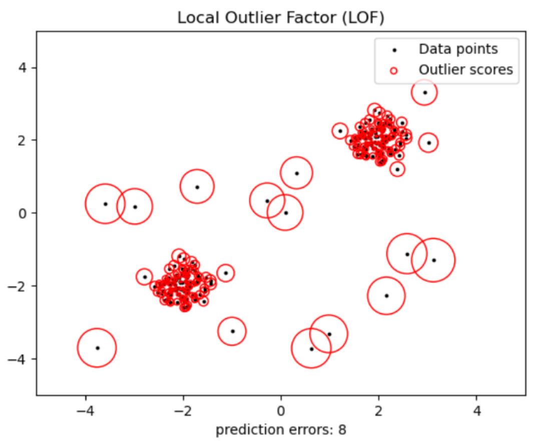


Figure 3.1: Graphical example of the outlier score of each point [18]

3.1.2 Isolation Forest

The Isolation Forest algorithm leverages the concept of binary trees and random partitioning to separate anomalies from the majority of data points effectively. It operates on the fundamental idea that anomalies are more "isolated" and require fewer splits in the data to be separated from the rest. In contrast, normal data points require more partitions to be isolated.
Therefore, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node. This path length, averaged over a forest of such random trees, is the anomaly score given to each point by the algorithm.

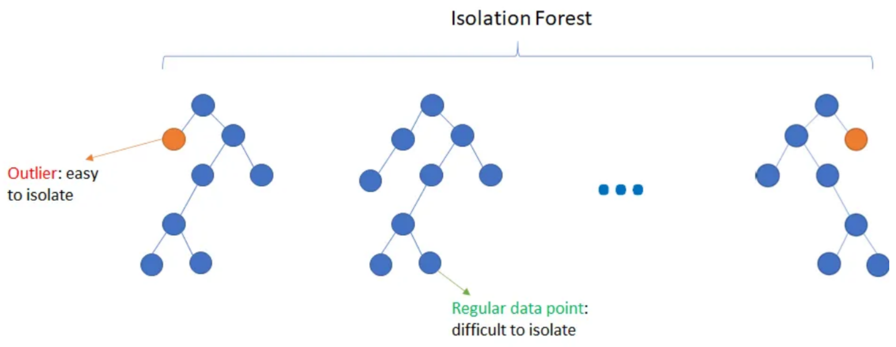


Figure 3.2: Graphical example of the early split done in the trees which separates the outliers [16]

3.1.3 One class SVM

One-Class SVM is based on the principles of support vector machines, a class of supervised learning algorithms used for classification and regression tasks. However, the One-Class SVM focuses on the unsupervised task of anomaly detection, specifically designed for situations where labeled anomalous data is scarce or non-existent. OCSVM attempts to find a hyperplane that maximizes the margin while containing as many normal data points as possible. With "margin" (see Fig.(3.3)), we mean the minimum distance between the samples of the classes, in our case two, separated. Data points that fall outside this margin are considered anomalies.

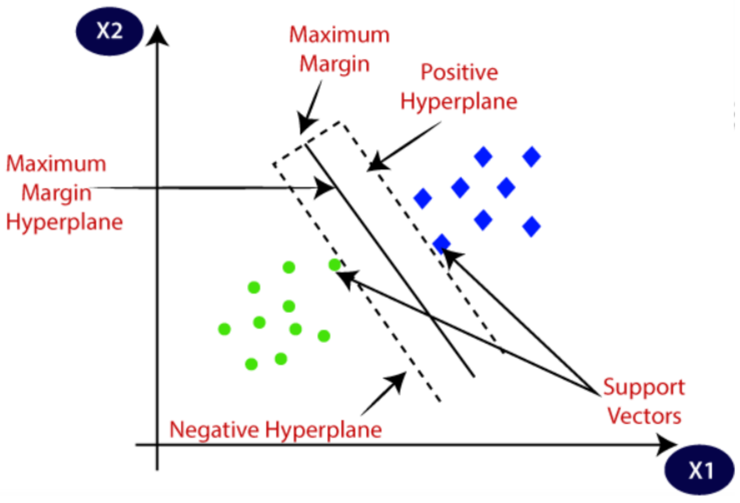


Figure 3.3: In the figure is shown an explanatory picture about the definition of margin for algorithms in svm's family. The support vector are used to slit the different classes [34]

3.1.4 Matrix Profile based method: DAMP

As mentioned in previous sections, time series analysis is a broad field that encompasses various sub-study paths, thanks to the versatility of this data modality.
This work began by delving into existing techniques and subsequently exploring new potential approaches for outlier analysis in time series data. One of the initial challenges encountered was the precise definition of "outlier" because, depending on the specific case, it can have different interpretations.
For starters, we have focused on one of Keogh et all. last work [44], in which the definition of "Discord" is given; to fully introduce this new object, we first need to define the concept of "Non-Self Match" in the TS context:

Definition 1 *Given a time series X, and one of its sub-sequences $x_{i:j}$ of length M starting at position i and a matching sub-sequence $x_{k:j}$ starting at k, then $x_{i:j}$ is a NON-self match to $x_{k:j}$ if the distance between i and k is such that:*

$$d_{ik} = |i - k| \geq M$$

Meanwhile with Discord, we intuitively associate a sub-sequence of the series which is inconsistent, or discordant, with the rest of it. To be more precise:

Definition 2 *Given a time series X and one of its sub-sequences $x_{i:j}$ of length M, is said to be a Discord for X, if the distance between $x_{i:j}$ and its nearest NON-SELF MATCH is maximum.*

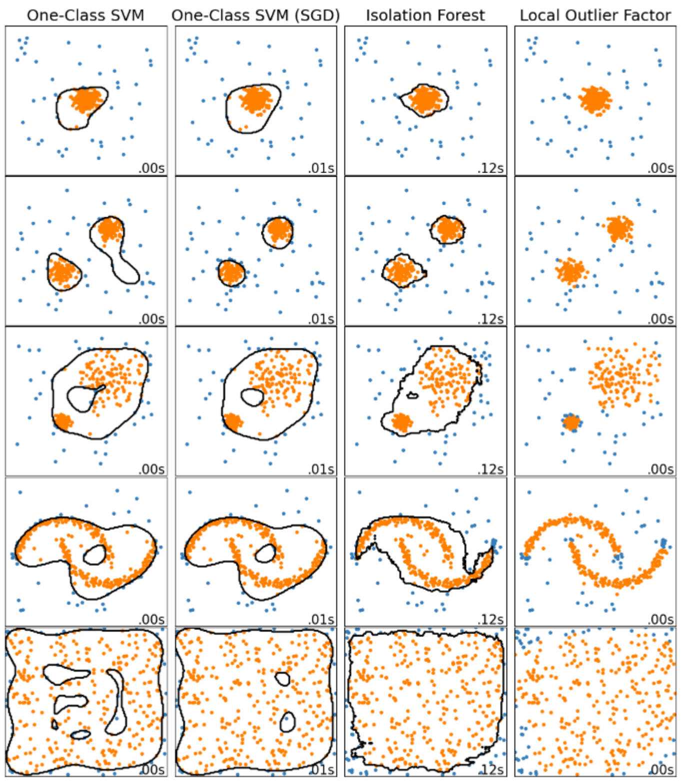


Figure 3.4: Comparison between the class bounds for five datasets, found by the different algorithms

In [44] the authors introduce DAMP, an algorithm which computes for each candidate query $x_{i:j}$ the distance between itself and every other non-self matching sub-sequence of the TS, and puts the minimum distance found in a vector called Matrix Profile. Therefore, finding the maximum value in the Matrix Profile leads to the predicted Discord for the TS X.

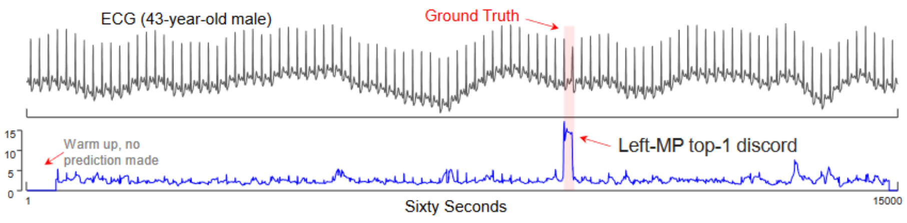


Figure 3.5: Example of the application of the matrix profile onto a time series recording an ECG. It can be seen that to the real discord sub-sequence is associated the maximum value in the matrix profile [44]

This kind of definition finds its applications mostly when dealing with TS in which unusual behaviours are recorded (e.g. an ECG that records the moments before and after an heart attack).
After a first focus on the possible applications of DAMP, we decided to move forward, shifting substantially the background of our work by associating outliers with the transitions between different states in TS. Because of this choice, the discord's case couldn't be applied in an efficient way, being that the kind of outliers we were searching for were intrinsically different: DAMP needs strong and sudden changes, ones which don't occur often, instead of gradual and recurring ones.
Therefore, in order to follow our new aim, it was necessary to find a new tool.

3.2 Time Series Segmentation and Transition recognition

Let us consider a multivariate a time series $X \in \mathbf{R^{N \times M}}$, where $M \geq 1$ stand for its number of channels while N for the number of samples. Moreover we suppose that X is such that some of its characteristics change abruptly at some unknown time indexes $1 < t_1 < t_2 ... < t_L < N$. This family of time indexes $\{t_i\}_{i=1}^{L}$, is called "change points" [48], "transitional points" or even "cut points". The need to identify these points has given rise to time series segmentation and transition recognition tasks. For the latter, it constitutes the primary objective of the problem, and for the former, it is a crucial step required to proceed with the analysis. Hence, there exists a strong connection between the two.
Depending on the context, the number L of changes may or may not be known. In cases where it is not known, it must also be estimated. In real-world scenarios, this information is often missing, particularly when domain-specific knowledge is lacking.
However, change point detection consists in choosing the best possible segmentation T according to a chosen loss function L(T,X) which must be minimized.

The task of cut point detection can be considered the cornerstone upon which transition recognition and time series segmentation are built.
Despite potential confusion, transition recognition, as we intend it in this work, goes a step beyond the sole change point detection task. It broadens the definition of cut points to include not only the indices at which the changes occur but also those immediately surrounding them. More on this will be explained in the following chapters.
Meanwhile, the segmentation task incorporates a classification step for the newly identified subsequences, with the aim of grouping them into sets with distinct characteristics.

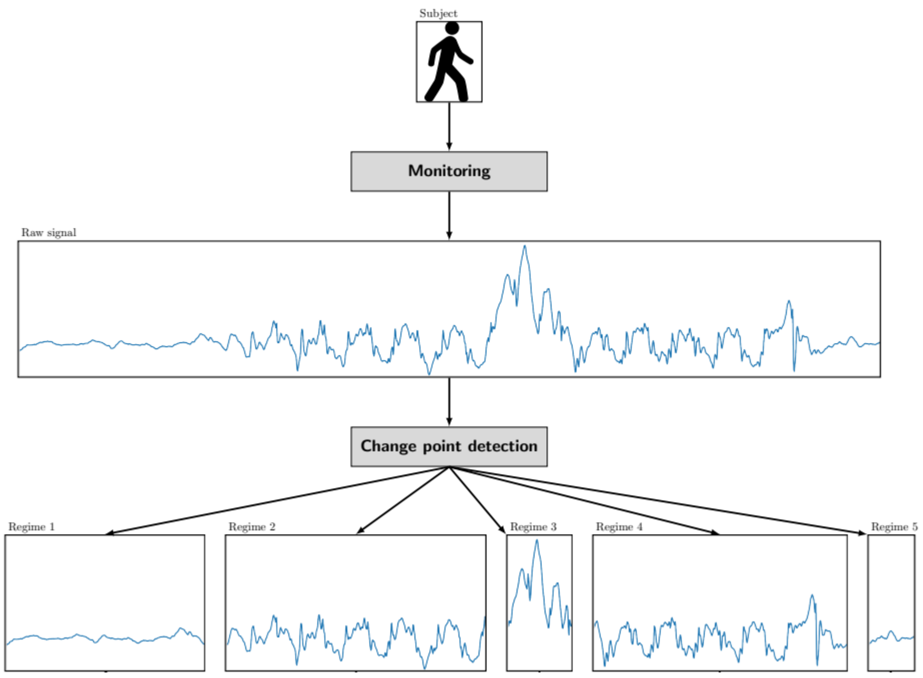


Figure 3.6: Simple graphical review of what change points are for the basic analysis [48]

3.2.1 Time2State

In June 2023, Chengyu Wang et al. released a paper titled "Time2State: An Unsupervised Framework for Inferring the Latent States in Time Series Data" [3]. In this paper, they introduced Time2State, a novel unsupervised algorithm capable of finding a satisfactory segmentation for time series data with minimal domain knowledge. In the field of time series segmentation, the objective is to identify

hidden patterns in the data that recur in certain segments of the series. We refer to these subsequences with similar features as 'states.' Thus, without any instructions regarding the number of possible states, their statistical characteristics, or any other information, Time2State can achieve such segmentation with a high level of accuracy for an unsupervised method.
Another feature that has elevated T2S to a state-of-the-art position in its field is its versatility, allowing it to be applied in various domains.

General outline of Time2State

The main challenges which the authors had to surmount in order to achieve their results were:

- Create and UNSUPERVISED and DOMAIN-FREE method: Clearly, the challenge of finding a solution that doesn't require prior knowledge about the data's characteristics, shape, and statistics is extremely demanding. Furthermore, the necessity of developing a solution applicable in any domain makes it even more formidable;
- No information about the number of classes: Since there is no specific bound on the number of states present, or at least possible, in the time series, the algorithm must adapt during its learning process and self-learn the parameters;
- Scalability: Given the large size of the data we work with nowadays, ensuring that the method is scalable is crucial;

The resulting algorithm itself contains two main phases: the training phase and detection phase.
During the first one, which we can see summarized in Fig.(3.7), an encoder is trained to learn the representation of the raw time series in an unsupervised and general manner. The main observations and assumptions used for this implementation are as follows:

- If we randomly select a sub-sequence from our time series, it is highly likely that other sequences nearby, particularly those obtained by sliding the initial window forward or backward with a step s, belong to the same class or state as the first one;
- If we randomly select multiple sub-sequences from our time series, it is highly probable that they belong to different states;

From these two observation, the authors introduced the Latent State Encoding Loss (LSE), which we will cover in the next paragraph.
As for the detection phase, which is summarized in Fig.(3.8), once the encoder

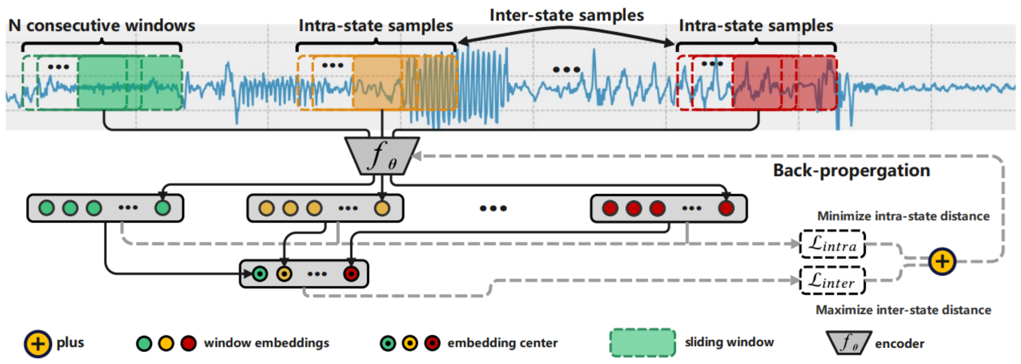


Figure 3.7: The training process of Time2State involves several rounds. In each training round, Time2State randomly selects N consecutive windows M times. It then improves the encoder by simultaneously reducing the overall distance between samples within the same state and increasing the overall distance between samples from different states [3]

is trained, a sliding window with a step size 's' traverses the entire time series. During this process, the embeddings for each window are computed, allowing the condensation of long series, often comprising multiple channels, into lower-dimensional representations.
Finally, the cluster structure of the transformed data in the embedded domain is identified, assigning each raw data point from the initial time series a label.

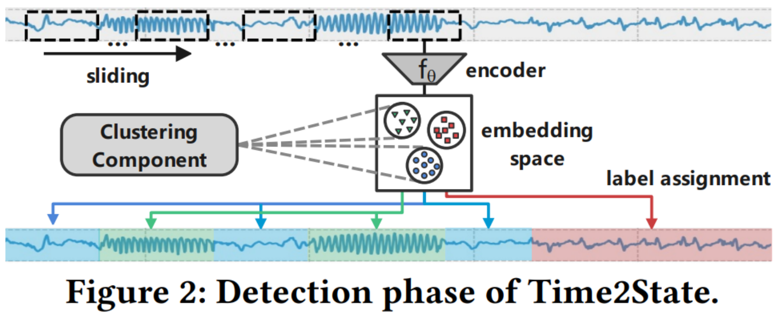


Figure 3.8: Explanatory figure of the detection phase of Time2State [3]

The LSE loss function

The Latent State Encoding relies on the assumption of data homogeneity, as we expect that points or windows that are temporally close to each other are likely to belong to the same state, while randomly selected ones are more likely to belong

to different states.
Thanks to this characteristic, it becomes possible to construct an encoder that maps our initial time series into a lower-dimensional embedded sequence using a sliding window approach. Therefore, the encoder itself is represented as $f_{enc} : \mathbf{R}^{\mathbf{w*d}} \rightarrow \mathbf{R}^{\mathbf{z}}$, where w, d and z correspond to the window size, channel number, and dimension of the embedding space, respectively.
With the newly transformed time series, we aim to identify a cluster structure in the data.
During the encoding process a two-stage sampling process unfolds:

- **INTER-STATE SAMPLING**: the algorithm chooses randomly M **non consecutive** windows of length w on the given TS. Their position is drawn from a uniform distribution U(0,T-w-N), where T is the length of the TS. The distribution is chosen such that the encoder can learn evenly the features of the entire series. To each of these windows a different state is assigned;
- **INTRA-STATE SAMPLING**: this second step takes on from the last one, since the algorithm extracts N samples for each of the previous states, sliding M windows chosen in the last step, forward for N-1 times with in a step size 1. For the assumptions introduced above, we will assign to these ones the same class as the window from which they were "originated";

After showing the two sampling stages, it is now possible to analyze the true LSE-Loss as:

$$\mathbf{L}_{LSE} = \mathbf{L}_{intra} + \mathbf{L}_{inter}.$$

Where, in particular:

$$\mathbf{L}_{intra} = \alpha_1 \sum_{k=1}^{M} \sum_{i=1}^{N} \sum_{j=1, j<i}^{N} -\log\Big(\sigma(f_{enc}(\mathbf{o}_i^k)^T f_{enc}(\mathbf{o}_j^k))\Big)$$

$$\mathbf{L}_{inter} = \alpha_2 \sum_{k=1}^{M} \sum_{l=1, l<k}^{M} -\log\Big(\sigma(-\mathbf{c}_k^t \mathbf{c}_l)\Big).$$

Regarding the various parameters in the two formulation we have:

- $\alpha_1 = \frac{2}{M*N*(N-1)}$, which is used to average the similarity;
- $\alpha_2 = \frac{2}{M*(M-1)}$, which is used to averaged the similarity;

- σ is the classic sigmoid function;
- $\mathbf{o}_i^k$ is the sample found by sliding the k-th window obtained in the intra sampling stage (so k<M) by i steps forward;
- The dot product $f_{enc}(x_p)^T f_{enc}(q)$ corresponds to the computation of the similarity [7]
- $c_k = \frac{1}{N}\sum_{i=1}^{N} f_{enc}(\mathbf{o}_i^k)$ is the embedding center. It is used to ease the computational cost of the similarity;

Both components of the LSE-Loss serve distinct purposes. The "intra" component aims to maximize the similarity between the representations of intra-state samples, assuming that they belong to the same state. This encourages their representations to be close in the embedding space.
Conversely, the "inter" component works to minimize the similarity between inter-state samples, assuming that they belong to different states. This pushes their representations apart in the embedding space.
It's evident that this procedure can lead to three possible outcomes: true intra-state samples, true inter-state samples, and false inter-state samples. This variability arises because randomly selected windows may or may not belong to different classes, and a sliding window might span different states. However, the true strength of this loss function lies in the interplay between $\mathbf{L}_{intra}$ and $\mathbf{L}_{inter}$. The latter aims to distance true inter-state samples, and in doing so, it won't increase the intra component. On the other hand, if it separates false inter-state samples $\mathbf{L}_{intra}$ will increase.
Hence $\mathbf{L}_{intra}$ is, in a way, a counter-force which permits the algorithm to focus on the true inter-state samples.

The Encoder

The selected encoder architecture is derived from the work "Unsupervised Scalable Representation Learning for Multivariate Time Series" by Jean-Yves Franceschi et all [42]. It consists of deep neural networks employing exponentially dilated causal convolutions.
In contrast to conventional recurrent neural networks, which have been tailored for sequential tasks, these networks are designed for scalability, enabling efficient parallelization on GPUs.
Exponentially dilated convolutions, in comparison to full convolutions, excel at capturing long-range dependencies at a constant depth. This is achieved by exponentially expanding the network's receptive field.
The model's structure is founded on stacks of dilated causal convolutions, which

map sequences to other sequences of the same length. Each output element i is computed using only the values up to the i-th index of the input. The "causal" aspect is evident in that, to compute the output for a given time t, only the information up to time t is utilized, without depending on future knowledge.
In-depth, the architecture comprises layers constructed as a combination of causal convolutions, weight normalizations, leaky ReLUs, and residual connections. Each of these layers is associated with an exponentially increasing dilation parameter, equal to 2^i for the i-th layer. Ultimately, the resulting output is passed through a global max-pooling layer that aggregates all temporal information into a fixed-size vector. The encoder's final output is a linear transformation of this vector, culminating in a user-defined fixed size that is independent of the initial size.
For a clearer understanding of the structure, refer to Fig.(3.9) which summarizes its different components.

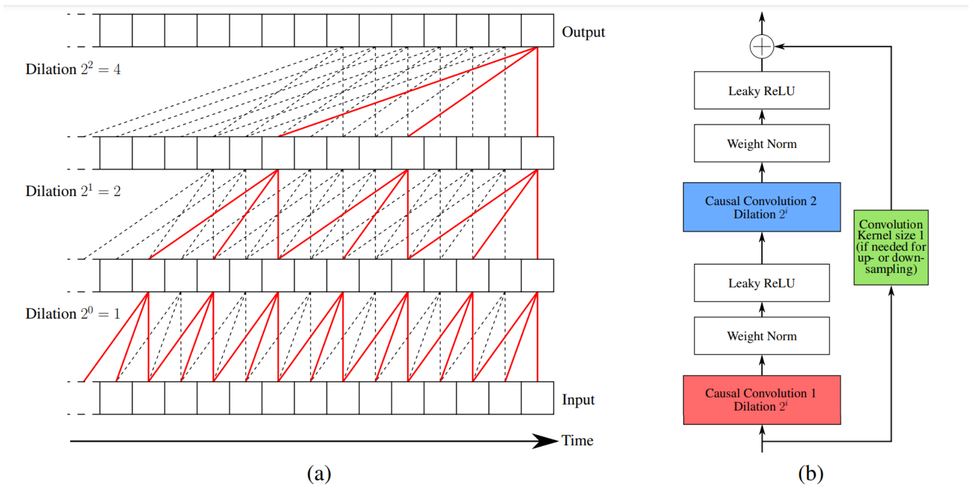

Output
Dilation $2^2 = 4$
Dilation $2^1 = 2$
Dilation $2^0 = 1$
Input
Time
(a)
Leaky ReLU
Weight Norm
Causal Convolution 2 Dilation 2^i
Leaky ReLU
Weight Norm
Causal Convolution 1 Dilation 2^i
Convolution Kernel size 1 (if needed for up- or down-sampling)
(b)

Figure 3.9: In the figure is shown the structure of the encoder. In (a) are shown three consecutively arranged dilated causal convolutions. The lines connecting each sequence represent their computational relationships. The solid red lines emphasize the dependency graph for calculating the final value of the output sequence, demonstrating that no information from future time points in the input time series is utilized in this computation. Meanwhile in (b) there's the composition of the i-th layer of the chosen architecture [42]

Embeddings visualization

In order to check if the whole process performed as foreseen, it is possible to visualize, setting an appropriate number of resulting channels for the embeddings, the outcome of the transformation of the TS.

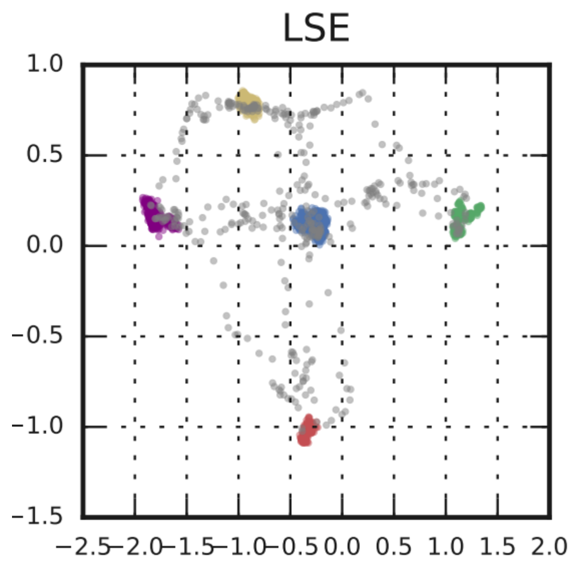


Figure 3.10: Embedding space learned by LSE-Loss. Points of the same color represents windows from the same state. Instead, the gray points stands for windows that span two states. Hence, the trajectories of gray points imply the transition between states [3]

From this plot, the main idea of this work was conceived: rather than concentrating solely on segmentation tasks, we shifted our attention towards the clustering or binary classification of individual data points within the series. The objective of this classification is to address the challenging task of identifying transitions between different states. Therefore, we define transition points as those samples situated between two distinct classes:

Definition 3 *Given a time series X and its set of possible states* $\mathbf{S}$*, being "a" the class associated to the observation* x_i*, if* x_{i-1} *belongs to the class "b", such that a is different from b, then* x_i *is a central transition point.*

For our purposes, in addition to identifying these "core" transition points, we also assign the same label to the t points preceding it and the t-1 points following it, where t is a specific parameter of our algorithm.
This choice is based on empirical knowledge that changes in state typically don't occur abruptly. To illustrate with an example, in the context of physical activities, if a subject is transitioning from walking to running, the change can be either gradual or more sudden, depending on the specific situation. Assigning the transitional class label solely to the central point would not only contradict common sense but would also make the task more challenging, as the central point and its closest neighbors are indeed quite similar.

Chapter 4

Presented approaches

4.1 General outline

The starting point of this work was to study and testing the capabilities of Time2State in tasks different from the segmentation, in order to test its performances. In particular, the segmentation task is strongly linked to the recognition of transition points, as defined in the previous chapter, since when a solution is found for the first task, we have an answer to the other one.
Technically speaking, if our segmentation were perfect, or of really great quality, we would have also have a good solution for the transition recognition. The true hurdle when facing not ideal solutions, is that for the segmentation task, our focus is on "big" portion of the series, hence since TS data in this context are usually long ones, at least 10^3 samples, getting a few entries wrong or delaying a bit the start of a new state, doesn't really hurt the general performance of the algorithm. Instead, when facing the binary classification of transitional/not transitional points, these kind of imprecision really weights both on the resulting recall, which is our most important performance measure, and precision of the algorithm. For tasks of this kind, with highly non-balanced classes, the focus is on recognizing the highest possible number of positive cases.
Therefore, we aim to have a high certainty about the validity on the points classified as positives (for us the transitional ones). In our case, being that we are dealing with sequential data, having the right information about even just one point, can give good clues about the other transition points, since, as we defined above, not only the "central" point between the change of two states is considered as transitional, but also t points before and t-1 after it.

4.2 Fully unsupervised approach

To find a meaningful competitor for our approach, we initially explored fully unsupervised methods. We briefly presented DAMP in the Introduction section of this work in [1]. However, after conducting experiments on transition recognition, we realized that the framework and the scope of its implementation were too different from ours, making it unsuitable for our purposes.
As a result, we narrowed down our direct competitor to Time2State.
Since it is designed to solve segmentation tasks, we tried to see how well it would perform in a linked area.
To adapt the algorithm to the transition recognition task, we followed the steps outlined in Fig.(4.1):

- Apply T2S to the full time series;
- Recognize and save the cut points between state changes;
- Label as "transitional", hence positives, t points before and t-1 points after such cut points;
- Label as "not transitional", hence negatives, all the other samples;
- Compute precision and recall considering this labelling as the predicted one;

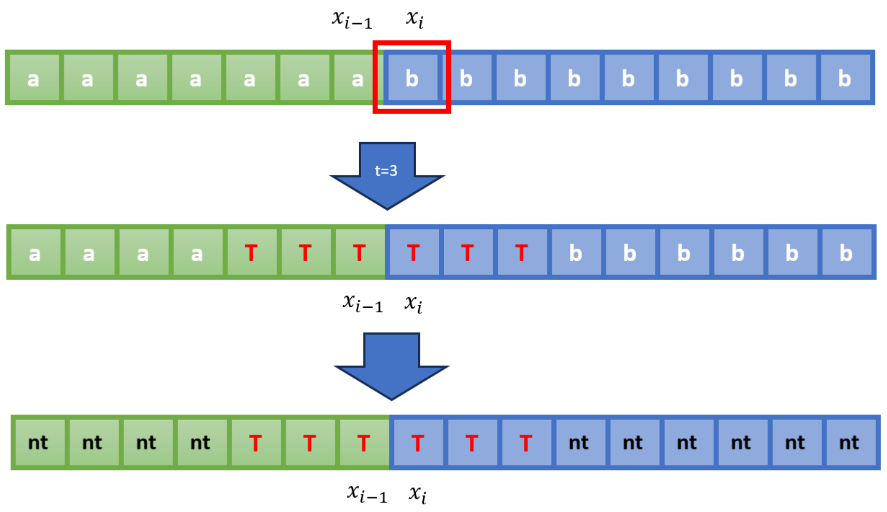

x_{i-1}
x_i
a a a a a a a b b b b b b b b b
t=3
a a a a T T T T T T b b b b b b
x_{i-1} x_i
nt nt nt nt T T T T T T nt nt nt nt nt nt
x_{i-1} x_i

Figure 4.1: Example of the extraction process of the new transitional class

We have chosen to compare our approach to this one mainly because if latter performed significantly well, adding restrictions relying to a semi-supervised method,

wouldn't have had any sense.
Therefore we have considered it a lower bound, or at least the minimum performances to achieve in order to find a meaningful algorithm.

4.3 The proposed semi-supervised approach

The input that inspired our approach is the visualization of the embeddings created by T2S during its data labeling procedure. Specifically, we hypothesized that if it were possible to create a cluster for each state detected by T2S, the transition points would ideally fall outside or somewhere in between these clusters. This is because those points share similarities with both the "departing" cluster and the "arriving" one, indicating that they are characterized by statistics and features that are close to, yet different from, both of the states between which the change is occurring.
Based on this supposition, we set out to find a measure that could emphasize or represent this condition. We primarily explored indices related to cohesion and the quality of clusters, such as the silhouette index [24] and separation, a submeasure used in silhouette.
In particular, we opted to work with the point-wise separation since it theoretically aligned perfectly with our problem. This measure yields a low value when the points we are considering have a minimal distance from a cluster to which they do not belong, indicating poor separation from the other set. Conversely, if the point in question is far from all other clusters to which it does not belong to, its separation value will be high. Therefore, if a point is a transitional one, as per our hypothesis, it should be located in the middle of two states, resulting in a lower-than-usual separation value.
With that explanation, the approach we have taken does not strictly adhere to the classical semi-supervised methodology because we never utilized ground-truth data during our process.
However, we have chosen to classify it as semi-supervised because it relies on the labels predicted by Time2States as "training data".
Hence, the pipeline followed, and summarized in Fig.(4.2) was:

- Taken a TS X, we split it into two, extracting a "train" and a "test" segment (it would be recommended that in the train part, the majority, or better, all the states which we could find in the TS are seen at least once);
- Apply Time2State to the training dataset extracting its labels. We will consider them as "semi-ground truth" for the next steps;
- Using as a basis the clustering on the training set, we assign the labels to the test set;

- Given the temporary labels, we compute the point-wise separation of the test samples;
- In order to recognize the transitional points, we sort the test in ascendant order with respect to the separation scores;
- Recognize and save the cut points between state changes with respect to the temporary label assigned before;
- Label as "transitional", hence positives, t points before and t-1 points after such cut points;
- Label as "not transitional", hence negatives, all the other samples;

At this point, the pipeline splits in different branches with respect to the experiment we have conducted.
Leaving out the ablation studies, the proposed approach continues as:

- Take the first k points in the new-sorted dataset;
- Apply unsupervised outliers detection algorithm, considering as outliers the transitions, and obtain the predicted labels
- Compute precision and recall using such labels

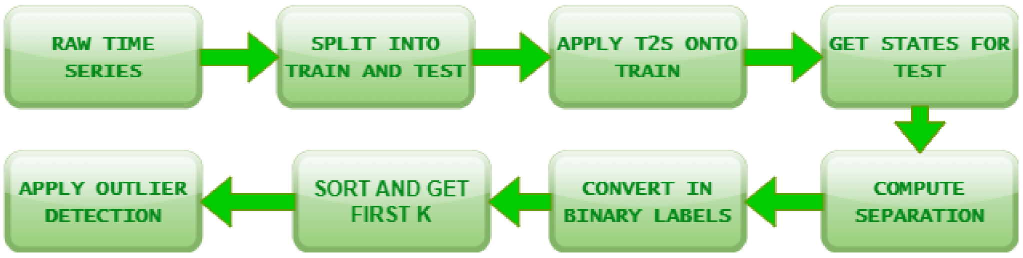


Figure 4.2: Summary of the first semi-supervised pipeline proposed

4.3.1 Feature engineering trough Shapelets Transform

In recent years, the field of time series analysis has witnessed significant advancements in techniques aimed at extracting meaningful patterns and features from time-ordered data. One such innovation that has gained considerable attention is the concept of "shapelets". Shapelets are defined in [43] as “subsequences that are in some sense maximally representative of a class”.
Intuitively, in the context of a binary classification setting, a shapelet is considered

discriminant if it is predominantly present in most samples of one class and consistently absent from sequences of the other class. To assess the degree of presence, when given a time series X, one employs shapelet matches:

$$d(x, s) = min_t||x_{t:t+L} - s||_2$$

where :

- s is the shapelet considered;
- L is the length of the shapelet s;
- $x_{t:t+L}$ is the subsequence of X, which goes from temporal index t to t+L;

If the distance, denoted as d(x, s) [43], is sufficiently small, we can infer the presence of the shapelet s in the time series or a closely related subsequence.
Therefore, shapelets can be defined as discriminative subseries or subsequences within a time series that capture essential characteristics and patterns, serving as representatives of the entire time series.
In a classification context, the objective is to identify the most significant shapelets based on labeled time series data. These shapelets can be either extracted from the training dataset or learned through gradient descent [43].
Moreover, shapelets have several useful characteristics:

- **Variable Length**: Unlike fixed-length subsequences, shapelets can have varying lengths to capture different patterns within the data. This adaptability is crucial for handling diverse and complex time series;
- **Position Invariance**: Shapelets are position-invariant, meaning that they can occur at different positions within a time series while still carrying the same discriminatory information.
- **Interpretability**: Shapelets provide a degree of interpretability to the modeling process. Researchers and data scientists can gain insights into which specific features or patterns are influential in the classification or clustering task.
- **Reduced Dimensionality**: Shapelets can reduce the dimensionality of time series data, making it computationally more efficient while preserving the most relevant information.

The concept of shapelets finds applications in a wide range of fields, including Medical Diagnostics, Environmental Monitoring, and even Financial Time Series Analysis [43].
In this study, we utilized this tool to restructure subsequences within the time series, achieving both dimensionality reduction and the extraction of a representation containing more information than the raw series data.
Specifically, after partitioning our data into training and test sets, we conducted a search for the top four shapelets of length G. The choice of G was made to be long enough to capture potential sequence behaviors while remaining shorter than the average duration of a state.
Subsequently, we adjusted the data granularity used for analysis. Instead of working with each individual time unit sample, we aggregated M of them into a subsequence. Following this transformation, we restructured each of these subsequences based on their distance from the four previously identified shapelets.
After this, the procedure is similar to the one followed by the previous section:

- Apply Time2State to the training dataset extracting its labels. We will consider them as "semi-ground truth" for the next steps;
- Using as a basis the clustering on the training set, we assign the labels to the test set;
- Given the temporary labels, we compute the point-wise separation of the test samples;
- In order to recognize the transitional points, we sort the test in ascendant order with respect to the separation scores;
- Recognize and save the cut points between state changes with respect to the temporary label assigned before;
- Label as "transitional", hence positives, t points before and t-1 points after such cut points;
- Label as "not transitional", hence negatives, all the other samples;
- Take the first k points in the new-sorted dataset;
- Apply unsupervised outliers detection algorithm (from the PYOD library), considering as outliers the transitions, and obtain the predicted labels;
- Compute precision and recall using such labels;

In Fig.(4.3) there whole pipeline is summarized.

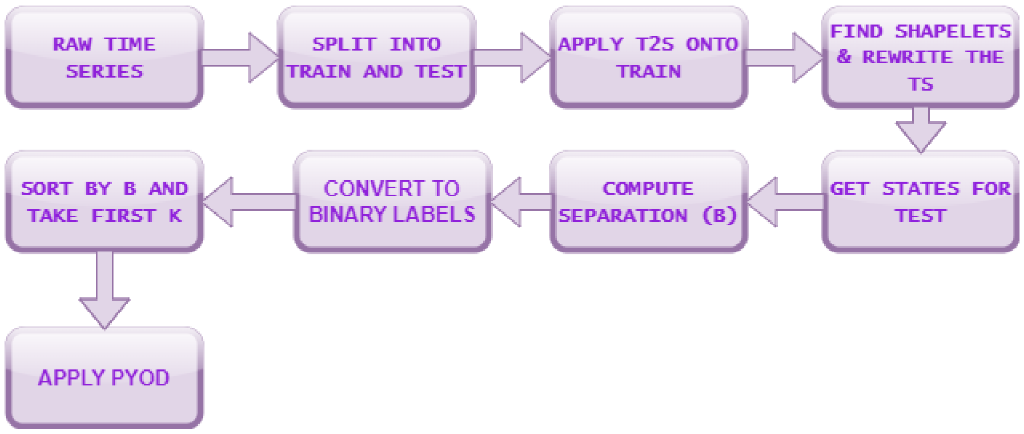


Figure 4.3: Summary of the second pipeline proposed, which employes shapelets transform

4.4 The supervised approach

As one might expect, we initially implemented a supervised approach to see how this more advanced techniques would perform. However, we soon discovered that simply applying algorithms such as Gradient Boosting [15] and Random Forest [23] was insufficient due to the complexity of the task at hand. Since our primary focus wasn't finding the best-suited supervised solution and we aimed for a method that required minimal fine-tuning and had a shorter computational time, we decided to explore alternative approaches.
The methods mentioned earlier faced time constraints, especially when applied to time series data with at least 10,000 entries. Consequently, we shifted our focus from the method itself to the "sampling strategy" employed. As discussed in the earlier sections, supervised methods excel in their ability to learn from the provided training data. In a scenario with highly imbalanced classes, randomly selecting a subset of the data for training could result in having no samples from the minority class.
In an attempt to address these issues, we opted for a simple Decision Tree [14] as the architectural choice and enhanced its performance with a customized train-test splitting technique.
Our approach consisted of the following steps:

- We started with the time series data, denoted as X, and split it based on its binary classes;
- Our objective was to create N sub-datasets from the original one, akin to the principles of a Bootstrap method;

- To maintain class imbalance, we applied extraction without replacement to both classes, generating N subgroups comprised of randomly selected samples from each class;
- We hence generate N subgroups putting together the samples extracted from both classes;
- Subsequently, we employed N-1 of these datasets as training sets for separate decision trees, effectively training N-1 trees at this stage;
- Afterward, we used the trained trees to predict the class of our test set, which corresponds to the Nth dataset generated;
- To make the final prediction, we conducted a majority vote across the N-1 trees for each sample;

It's essential to note that we did not perform any grid search on the trees but solely applied this bootstrap-like procedure, resulting in a significant improvement in our results compared to the previous two methods.
The graph below summarizes our pipeline:

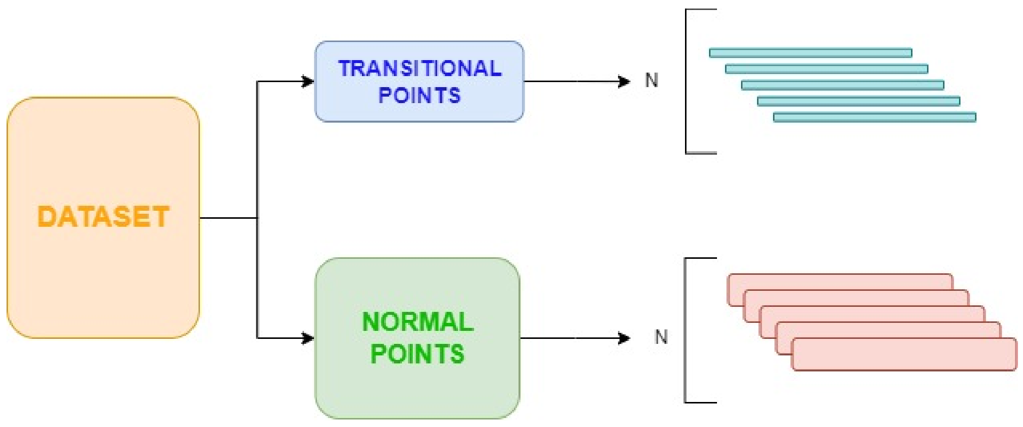


Figure 4.4: First part of the pipeline regarding the new sampling strategy tested

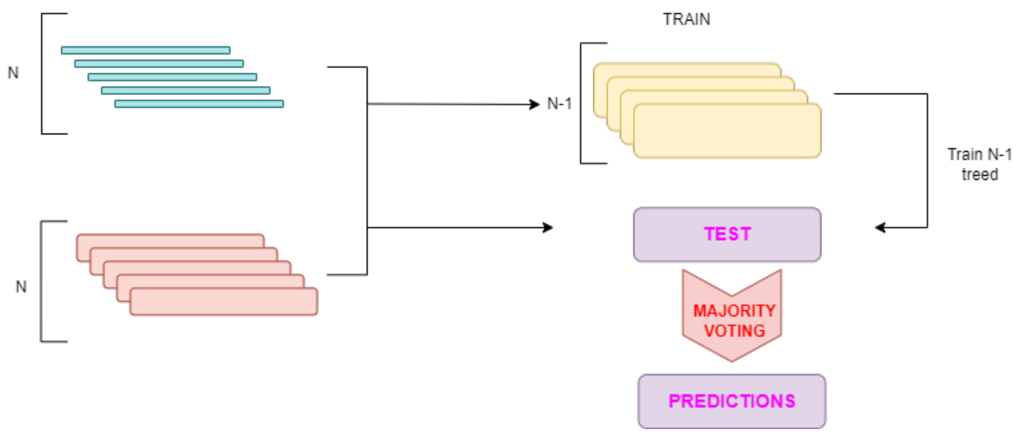


Figure 4.5: Second and final part of the sampling pipeline

Chapter 5

Experiments' design

Our experiments were organized with the primary goal of demonstrating the necessity of an upgrade concerning Time2State. To achieve this, we initially focused on identifying a well-performing and time-efficient supervised pipeline. Subsequently, we collected the performance results obtained with Time2State. Finally, we tested new approaches and conducted ablation studies.
In the following section, we will introduce the experimental design we have chosen for this work in greater detail

5.1 The Design

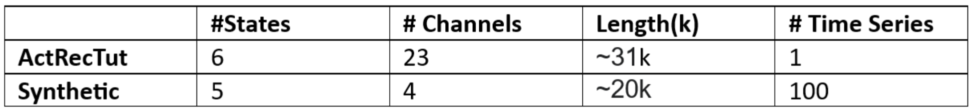
5.1.1 Datasets

One of the most challenging obstacles that we had to overcome during our work was, and still is, finding good and suitable datasets for testing our algorithms. Since the "normal" or usual tasks in the field of time series analysis are classification and regression, most datasets are generated or collected to satisfy their requirements
When dealing with segmentation tasks, we need longer time series that also exhibit different and recognizable patterns. However, this requirement can be easily satisfied by combining different shorter time series from the same collection. For example, the UCR Time Series Classification Archive [35] has a rich collection of short time series recording various phenomena (e.g., yoga, sports activities, etc.), which can be concatenated creating a "longer" time series.
In particular, this solution also addresses another crucial problem: having the "right" ground truth. Without careful consideration, it may not seem like a significant issue, but finding datasets with a clear and certified indication of what state we are in point-wise can't be taken for granted. For example, when linking different time series, we can simply use the class of the whole series as the one we are in

during its duration.
However, in our case, we are not merely interested in the succession of the states, but we are focusing on HOW the transition is happening.
For this reason, we had to discard many good and rich datasets that are typically used.
Unfortunately, we could only rely on two datasets, which were taken from the ones used for Time2State:

- ActRectut [30]: This dataset contains acceleration data collected from sports activities. For our purposes, the only suitable dataset of the collection was the one regarding the subject's walking. As said before, this dataset wasn't made for this purpose, but for mining frequent patterns, and hence there isn't enough labelled data regarding other activities;

- Synthetic [2]: In order to create this collection of 100 datasets, the authors used TSAGen [2] to generate synthetic time series, each containing 5 states. Moreover, the TS' states have random duration and succession pattern;



	#States	# Channels	Length(k)	# Time Series
ActRecTut	6	23	~31k	1
Synthetic	5	4	~20k	100

Figure 5.1: Table containing the specifics of the dataset used for the analysis

5.1.2 Ground Truth

As mentioned above, we ultimately decided on using for the testing procedure a combination of synthetic datasets and a real world one, called, ActRecTut [30]. Specifically, concerning the synthetic datasets, we conducted our tests on the entire set of 100 datasets. The results for our procedure were determined by calculating the mean and the standard deviation of the outputs to assess their variability.
We invested considerable effort into exploring different stylistic choices to determine the most appropriate temporal granularity for our data and, consequently, how to best assign its ground truth values. This step was crucial in our workflow since we were adapting datasets originally designed for a specific task to a different one, which inherently presents differences.
We explored two approaches, one involving data aggregation in a sliding window fashion and the other retaining the data in its original form. Depending on the case, we followed two different methods for labeling the data, always taking the

Ground Truth provided by the original authors as our starting point for these transformations:

- **POINT-WISE DATA**: In this scenario, we made minimal alterations to the original labels. However, we introduced the concept of "transitional points." As a result, we established an additional state to represent those points located in the midst of a label change. Consequently, we assigned this new label to the t points before and after a state change:

  Definition 4 *Given a time series X, indexed by time instant t, if x_{i-1} belongs to the class A and x_i to the class B, with A=!B, i is the time index of a "cut point". To all the points $x_{i-t:i+t}$ will be assigned the class "transitional".*

  In Figure(5.2) we show a graphic example of this procedure by plotting a piece of ActRecTut dataset, first channel.

- **WINDOWED DATA**: In this approach, we opted to restructure the data into sliding windows, leveraging the Shapelets transform technique as previously introduced. By doing so, we accomplished both dimensionality and feature reduction. This is achieved through data aggregation (dimensionality reduction) and expressing the new samples based on their distance from the K most significant shapelets in the training set (feature reduction).
  To elaborate, we initially divided our dataset into windows, each of a fixed length M, with a stride of T. Subsequently, we split the dataset into training and testing subsets. Then, we extracted the top K shapelets of length G from the training data (where G must be shorter than M to effectively capture recurring shapes in the subsequences). Finally, we have transformed the samples using as new features the distance between the sample itself and each of the shapelets found.
  This transformation was applied to both the training and testing datasets, following the guidelines outlined in the tslearn documentation [33].
  We generated the new ground truth (New_GT) by adjusting the original ground truth (GT). In this arrangement, when the last P points of a window belong to class X, the entire window is assigned to class X:

  Definition 5 *Given a time series X, indexed by time instant t, and one of its windows $q = x_{i:i+M}$, the class which will be assigned to q, will only depend on the label associated to its final P points. Hence the new ground truth assigned to the data will follow the rule:*

$$New\_GT = GT_{M-P:M-P+\#Windows}$$

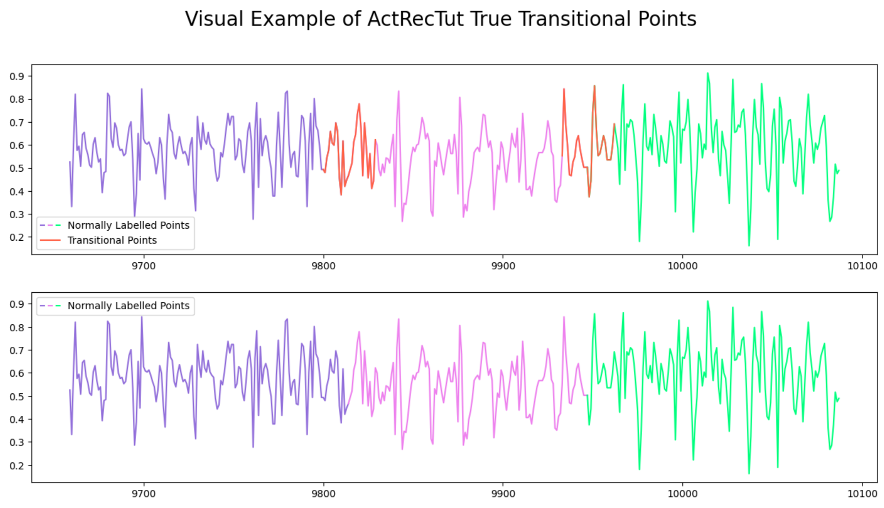


Figure 5.2: The two plots show a segment of ActRecTut data from its first channel. In the one below, we have colored each point depending on its state, meanwhile, in the other one we have added the new transitional points, extracted from the GT, and highlighted them in red.

Figure(5.3) shows graphically the last procedure, considering M = 5 , P = 2 and len(Dataset) = 20 (we can find the number of windows as #Windows = len(Dataset) - M +1).

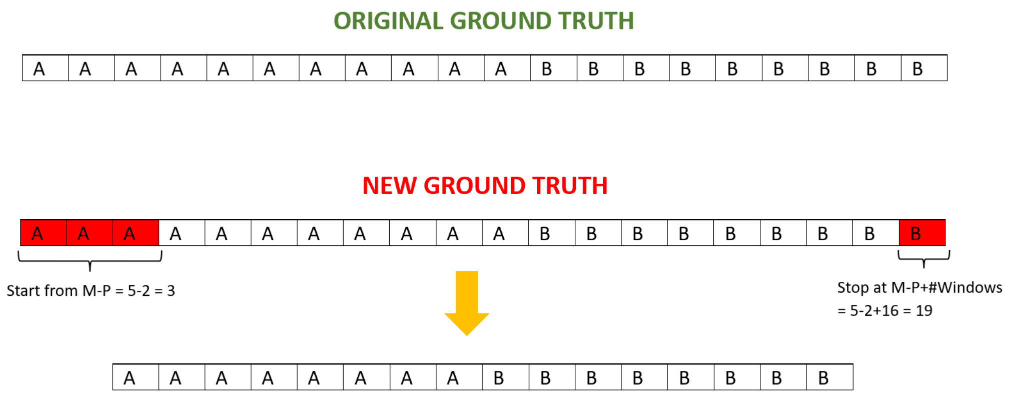

ORIGINAL GROUND TRUTH
A A A A A A A A A A A B B B B B B B B B
NEW GROUND TRUTH
A A A A A A A A A A A B B B B B B B B B
Start from M-P = 5-2 = 3
Stop at M-P+#Windows = 5-2+16 = 19
A A A A A A A A B B B B B B B B

Figure 5.3: Graphical example of the criteria for the new ground truth

5.1.3 Features and Hyperparameters

In addition to the Ground Truth discussion mentioned earlier, the features considered in the experiments also varied depending on whether we employed the windowed data or point-wise data approach.
When using the original data granularity, we retained the features or channels present in the data. To these, we introduced a unique element of our approach: the "b_score"[36].
The "b_score" for a particular data point is computed as the minimum distance between that point and a cluster different from the one it belongs to. The goal was to demonstrate that this metric provides valuable information for our task. When a sample point belongs to the transitional class, it implies significant similarities with both the "leaving" and "arriving" sets. Ideally, we aim to have distinct and separate label sets with transition points in between, as illustrated in Fig.(5.5).
In an ideal dataset scenario, where labels are well separated, as in Fig.(5.4), the "b_score" (also known as separation) of these samples would be significantly lower than that of regular samples. Thus, it would serve as an optimal, cost-effective distinctive feature.
In the case of the windowed approach, as previously mentioned in the introduction of the Shapelets transform method, we chose to restructure all samples based on the top K shapelets identified in the training set. In addition to these three new column features, we also included the "b_score," which was calculated after the transformation, as it led to a slight improvement in the results.

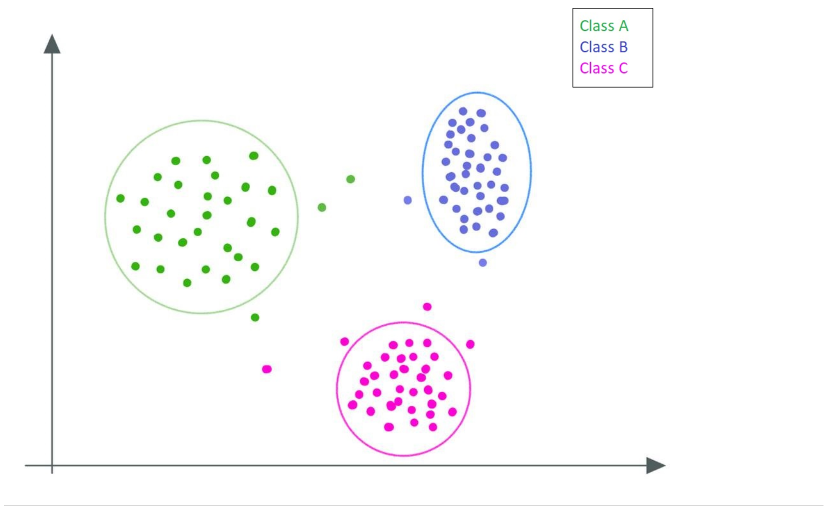


Figure 5.4: Example of ideal-case scenario. Here the three classes are well separated, except for a few samples which stands in between the sets. These ones represents the ideal-transitional points.

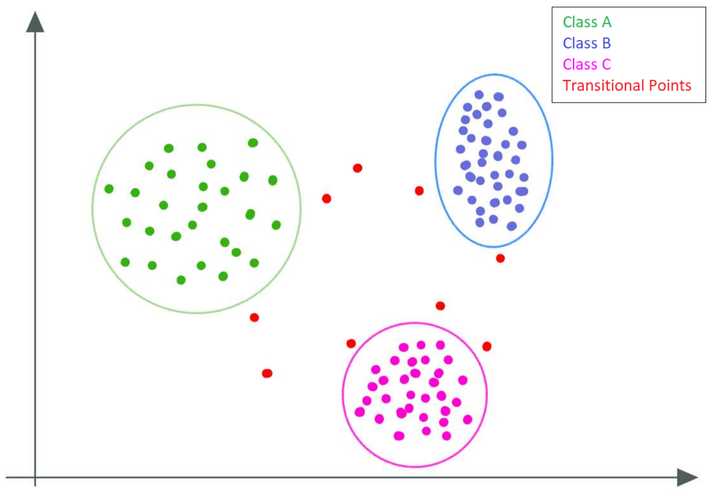


Figure 5.5: Follow up scenario with respect to the previous figure. In here the transitional points are set as the forth class.

Regarding the hyperparameters used in our various experiments, we made choices to fix some while fine-tuning others. Specifically:

- **Length of the subsequences** (M): We set this parameter to 100, achieving a balance between the average length of states and the obtained results;
- **Duration of transition**: The choice of this parameter depends on the selected modality, whether windowed or not. For the former, we fixed it at 15, resulting in a window of 30 entries, after also considering 10 and 20 as options. For the latter, we set it to 20, while also testing 15. These selections strike a balance between the logical definition of transition, which needs to be a fraction or significantly smaller than the average state duration, and the results obtained;
- **Length of the shapelets**: This parameter's value is based on the length of the transition. We aim to recognize shapes that are more significant in terms of duration compared to rapid state changes but shorter than the subsequences we are considering. Therefore, we set this parameter to 40 (equivalent to Duration of Transition $\times$ 2);
- **Number of top shapelets to consider**: We decided to fix this parameter at 4 to achieve feature reduction while retaining a substantial amount of information;
- **Shifting parameter for GT** (P): We chose to fix this parameter at 3, after considering other options. This parameter represents the "delay" permitted for the algorithm before recognizing the transition of a window to the new class;

- **Algorithms hyperparameters**: Details of the hyperparameters for each algorithm will be discussed in the following chapter, where we introduce the individual algorithms used;
- **Top K samples**, with respect to *b_score*: We will present the best results obtained after fine-tuning this parameter, which was adjusted within the range of 80 to 300;

5.2 Explaining Experiments

In order to better understand the experiments' results which will be shown in the next chapter, we dedicate this section to deeper introduce our methods.

5.2.1 Supervised

For the supervised case, we compared three different approaches, without digging deeper into the fine tuning of the models, since this background wasn't our focus. Initially we tried out two plain techniques: Random Forest and Gradient Boosting. We have decided to directly applying them doing a rapid grid search on the parameters:

- **Max_Features**: The number of features to consider when looking for the best split. This parameter changed with respect to the size of the dataset, since we had a large disparity between ActRecTut and the Synthetic ones. We tried out [1,2,3,4] for the latter, and [1,3,5,7] for the former;
- **Max_Depth** : Indicates the maximum number of nodes in the tree. We tried out [4,5,6,7];
- **N_estimators**: The number of boosting stages to perform or the number of trees in the forest. We fixed this parameter to 500;

In addition to these two methods, we also explored an ad-hoc ensemble method that incorporates a specific sampling strategy.
The sampling strategy aims to address the challenge posed by working with an exceptionally unbalanced dataset. While the datasets themselves are considerably longer than usual, the occurrences of state changes are relatively infrequent. As a result, the number of transitional points is notably lower compared to the "normal" points. Consequently, there's a risk of not having a sufficient number of transitional points in the training set for our algorithm to effectively "learn" to recognize and distinguish them.
Inspired by bootstrapping methods, we chose to implement an alternative sampling technique, which we describe below:

- Given the whole dataset X, already made binary in its labels, find and store in out_idx all the indexes of the outlier samples;
- Drawing without replacement we split into two equal parts out_idx, generating out_train indices and out_test indices;

- We compute the number of training set that we will generate as $Num\_sets = \frac{\#normal\_samples}{\#outlier\_samples}$;
- Then we repeat the following *FOR* $Num\_sets - 1$ times:
  - Draw without replacement N normal points, where N = # outlier data samples;
  - Add to these points the outliers in the training set;
  - Add the new dataset into a the list Train_Datasets;
- After this loop, we create the test set stacking the remaining normal points and the test outliers;

Having obtained the new train and test set, we proceeded to the learning step of the our method. In particular we employed a simple classification tree as pivot for the architecture.
As for the previous two methods, we tried out the same parameters, except for N_estimators, since we didn't require it.
The pipeline continues as:

- We repeat the following for $Num\_sets - 1$ times:
  - Train a classification tree using as training set the i-th dataset of Train_Dataset;
  - Test the tree on the test set extracted during the sampling stage;
  - Store the predicted classes in the matrix Vote;
- At the end of the loop, apply a majority voting on the matrix Vote, in order to obtain the final predictions;

5.2.2 Unsupervised

Under the unsupervised case, we put put what could also be called the ablation studies we conducted, since we tested out various methods applied on their own. Among those methods we have:

- **Time2state**
- **Local Outlier Factor**
- **Isolation Forest**
- **One-Class SVM**

All of them had been directly applied on the dataset, without any previous intervention.
For the last three, we have done a two parameter grid-search since we didn't have any domain knowledge and wanted to actually get the best result possible, considering also the computational time required:

- **Contamination**: This parameter is common to all the three methods. It represents the proportion of outliers in the data set. We have tried setting it to [0.01 : 0.15] with a step of 0.01;
- **n_neighbors** (LOF) : Number of neighbors to use t for k neighbors queries. We have tried setting it to [20 : 70] with a step of 5;
- **n_estimators** (IOF) : The number of base estimators in the ensemble. We have tried setting it to [100, 200, 500, 1000];
- **nu** (OCSVM) : An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors. Should be in the interval (0, 1]. We have tried setting it to [0.1 : 0.5] with a step of 0.07;

Regarding Time2State, we adhered to the standard choices made by the authors. Unlike the previous three methods, in this case, we do not receive only a binary classification as output but a state segmentation. Therefore, to obtain the prediction, we had to convert it into a binary format, using the same procedure employed to derive the new ground truth.
In Figure (5.6) it is shown an example of the ground truth plotting the first channel of ActRecTut, while in Figure (5.7) the results obtained applying Time2State. Meanwhile in the following two plots in Fig.(5.8, we show the same situation but with a zoom in;

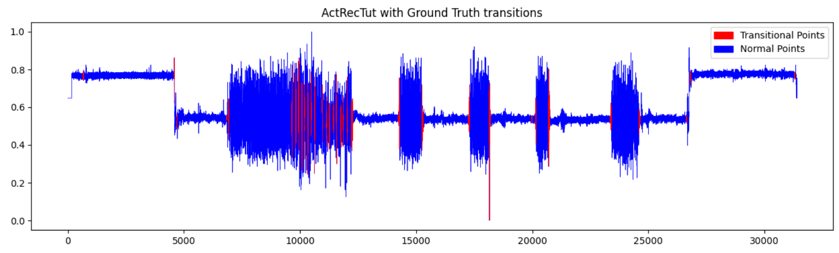


Figure 5.6: Plot of the first channel in ActRecTut. The blue points are the "normal" ones, while in red are highlighted the transitional points following the ground truth

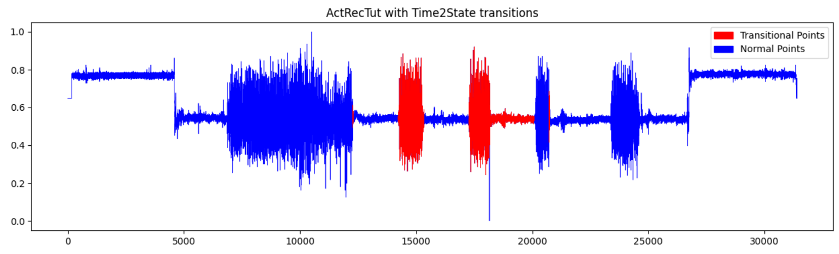


Figure 5.7: Plot of the first channel in ActRecTut. The blue points are the "normal" ones, while in red are highlighted the transitional points predicted by Time2State. Comparing the result with the previous figure, the difference catches the eye.

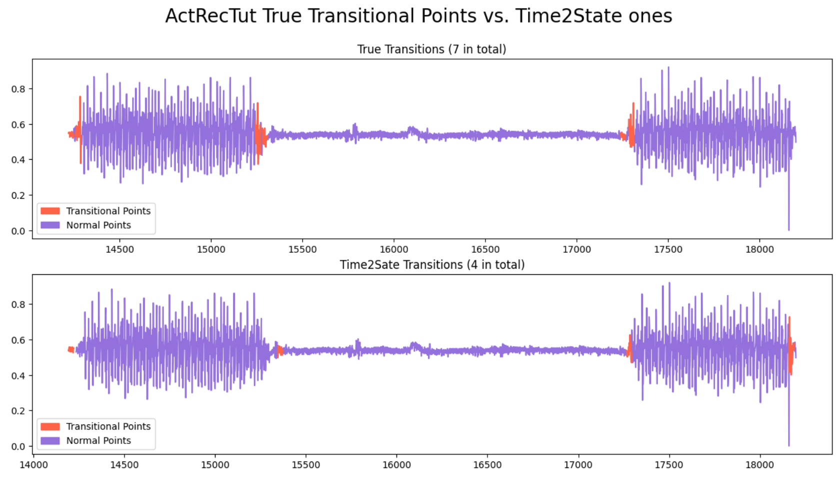


Figure 5.8: Plot of a section of the first channel in ActRecTut. With this figure it is possible to notice not only the difference in the position of the outliers predicted by Time2State, but also in the quantity.

5.2.3 Semi-supervised

Within the semi-supervised section, we have introduced the "new" methodologies we explored with the aim of enhancing the unsupervised approaches.
A common element across all implemented pipelines is the application of unsupervised outlier recognition methods, all of which are sourced from the PYOD library [32]. This choice is driven by the fact that transitional points can be considered outliers due to their deviation from the distribution of "normal" data, and the limited number of samples available to form a distinct set.
The initial set of experiments is centered around point-wise data.
In this context, we propose a semi-supervised approach that relies not on the ground truth but on the labels predicted by Time2State. Consequently, this method is more unsupervised than semi-supervised, although it underscores the necessity of utilizing a classification for a portion of the series.
The steps we took are the following:

- Given the original dataset X, we applied Time2State;
- Then we split the dataset into train and test, following an 80 to 20 fashion;
- For the purpose of computing the b_score later in the pipeline, we assign temporary labels to the test points by doing a majority voting between their top 10 neighbours from the training set. We stress that this process can't give us information about the transition, since we have lost the temporal order of the test samples;
- We compute the point-wise b_score with respect to the labels assigned before, and without considering the training data;
- Then we sort the samples with respect to the b_score in ascending order, so that the first points are the ones that are nearest to a cluster different of their own (hence, hopefully the transitional points);
- We select the top K samples and apply on them LOF or IOF, in order to get the result;

For what concerns the hyperaramter in this scenario we have:

- **K**: is the number of samples we select after the sorting, we tried setting it to $[80 : 300]$ with a step of 10;
- **Contamination**: introduced in the unsupervised section. We tried setting it to $[0.01 : 0.2]$ with a step of 0.01;

- **n_neighbors** (LOF): introduced in the unsupervised section. We have tried setting it to [15 : 50] with a step of 1;
- **n_estimators** (IOF): introduced in the unsupervised section. We have tried setting it to [100, 200, 500, 1000];

Meanwhile in the case of windowed data the pipeline in slightly different, due to the introduction of the shapelets.
We decided to take on also these set of experiments since we didn't have any domain knowledge about the transition classification background, and hence it wasn't clear which choice would fit better our model.
The step taken are the following:

- Given the original dataset X, similarly to what we have done in the previous case, we applied Time2State in order to get the class labels;
- We split the dataset in a half, extracting train and test. The proportion is different from the previous one since aggregating the data would decrease the test population;
- We have aggregated both the test and the train into window form, of length M, and sliding step 1;
- Then we have computed the four most significant shapelets in the training set, and rewritten it with respect to the distance between each of this shapelets found, using as metric the one introduced in 4.3.1;
- As in the previous experiment, we assigned temporary classes to our sample, using a majority vote between the first ten nearest neighbours of each window;
- Then we computed the window-wise b_score and sorted the data with respect to it;
- Finally we took the first 300 samples, and applied LOF onto them, in order to get the result;

Once more, for what concerns the hyperparameter in this scenario we only have the one linked to LOF, since the other have been already shared in the introduction of this chapter:

- **Contamination**: We tried setting it to [0.001 : 0.2] with a step of 0.005;
- **n_neighbors**: We have tried setting it to [30 : 70] with a step of 5;

Chapter 6

Experiments

In this section, we will report the results obtained during the testing of the pipeline previously introduced.
For some of the experiments, since the grid search on the parameters would lead to a prohibitive number of plots, only the best-case scenario is shown, with some of the hyperparameters fixed. The parameter chosen will be stated in the title of the plot.

6.1 Unsupervised

In the first part of the testing session, we aimed to determine the performance of the Time2State and Outlier Detection algorithms from the PYOD library when used to recognize transitions between segments characterized by homogeneous trends.
Particularly, the Time2State case was the most important, as this entire work is based on the assumption that the algorithm cannot fully adapt to this task as it currently stands.
As mentioned in previous sections, one of the primary reasons this task is considered challenging is the nature of the transition points. These points reside between changes of state and cannot be regarded as simple outliers. They stand out due to their dual-state characteristics, making their categorization difficult. However, this is primarily valid for the core points where the true transitions occur. On the other hand, the points at the edges of the transitional interval (at the start and end) are likely more similar to the departing/arriving state than the other one.
In Figures (6.1) and (6.2), both the recall and precision obtained using Time2State on ActRecTut and the synthetic datasets are plotted. The results clearly illustrate the task's difficulty. Another important observation is the high standard deviation associated with the synthetic dataset results.
In conclusion, these figures are sufficient to justify the need to explore alternative

approaches to address this task, as we can confirm that Time2State is not suitable for solving it.

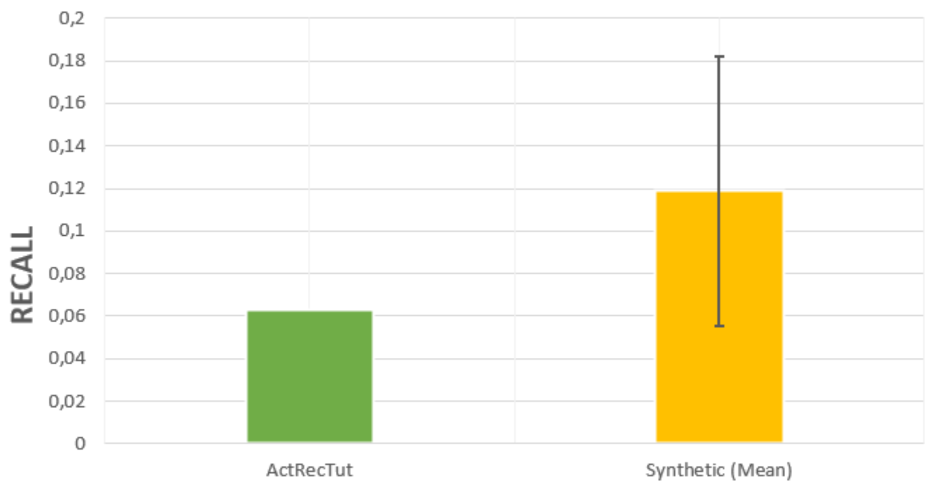


Figure 6.1: Resulting recall obtained by applying Time2State to ActRecTut and Synthetic datasets. For the latter the mean is shown, while the segment in black signals the standard deviation associated.

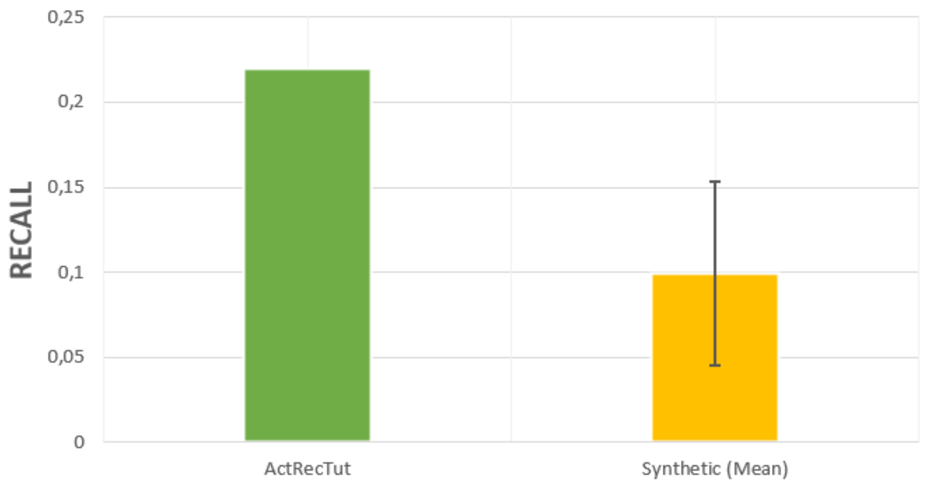


Figure 6.2: Resulting precision obtained by applying Time2State to ActRecTut and Synthetic datasets. For the latter the mean is shown, while the segment in black signals the standard deviation associated.

Meanwhile, for PYOD algorithms, the situation is slightly more favorable for

ActRecTut but deteriorates dramatically for the synthetic datasets.
The best performer between the PYOD algorithms tried is undoubtedly the Local Outlier Factor (Fig.(6.3) and Fig.(6.4)), even though there remains a significant performance gap between the two datasets. Furthermore, the computational time required to process the entire series using LOF was less than half an hour, making it the fastest among the outlier detection methods.
However, in general, it appears that all the approaches have difficulty identifying transitional points in the synthetic datasets. We suspect that this substantial difference in performance is due to the significant gap in the number of state transitions occurring in the two cases, with an average of 14 for the synthetic dataset compared to 44 for ActRecTut.

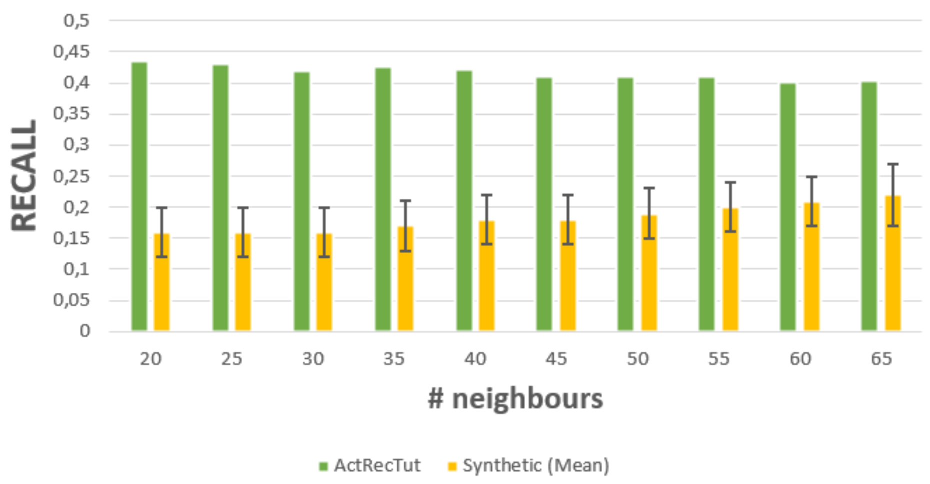


Figure 6.3: Resulting recall obtained by applying Local Outlier Factor to ActRecTut and Synthetic datasets, fixing contamination = 0.14. For the latter the mean is shown, while the segment in black signals the standard deviation associated.

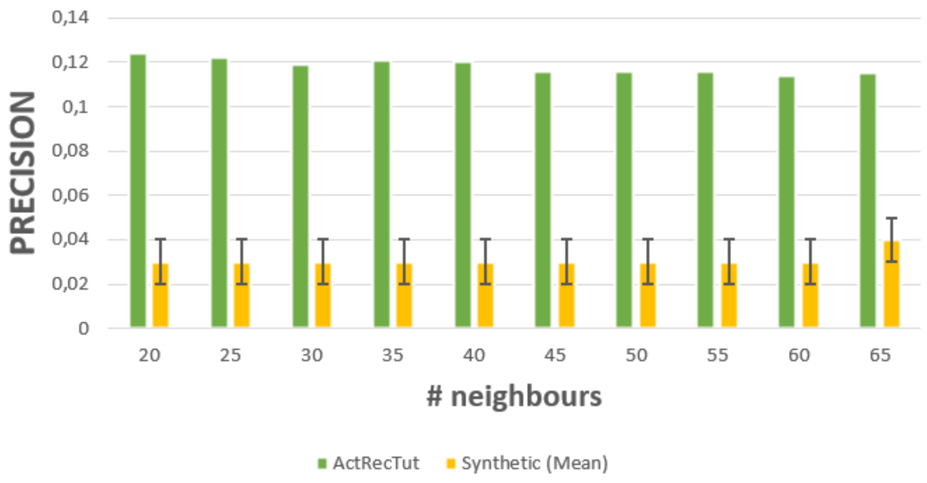


Figure 6.4: Resulting precision obtained by applying Local Outlier Factor to ActRecTut and Synthetic datasets, fixing contamination = 0.14. For the latter the mean is shown, while the segment in black signals the standard deviation associated.

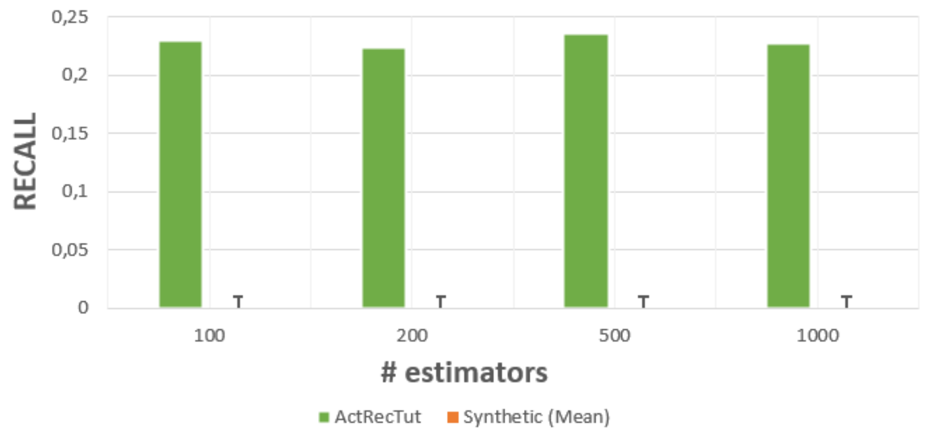


Figure 6.5: Resulting recall obtained by applying Isolation Forest to ActRecTut and Synthetic datasets, fixing contamination = 0.14. For the latter the result was a blank mean of 0 with only a minimum standard deviation associated.

As for Isolation Forest (see Fig.(6.6) and Fig.(6.5)) and One-class SVM (see Fig.(6.8) and Fig.(6.7), with a particular emphasis on the latter, not only are the results significantly worse than those of LOF, but the computational time increases to over one hour for OCSVM. Consequently, we excluded the latter from the other pipelines, both due to time constraints and the poor results we obtained.

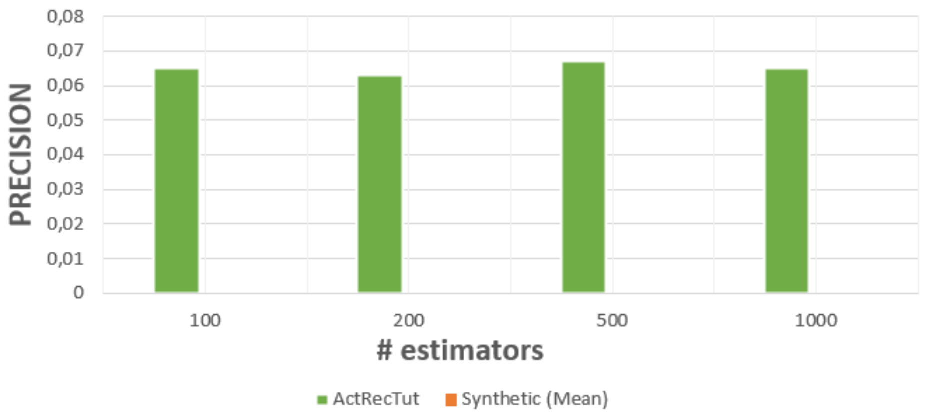


Figure 6.6: Resulting precision obtained by applying Isolation Forest to ActRecTut and Synthetic datasets, fixing contamination = 0.14. For the latter the result was a blank mean and standard deviation of 0.

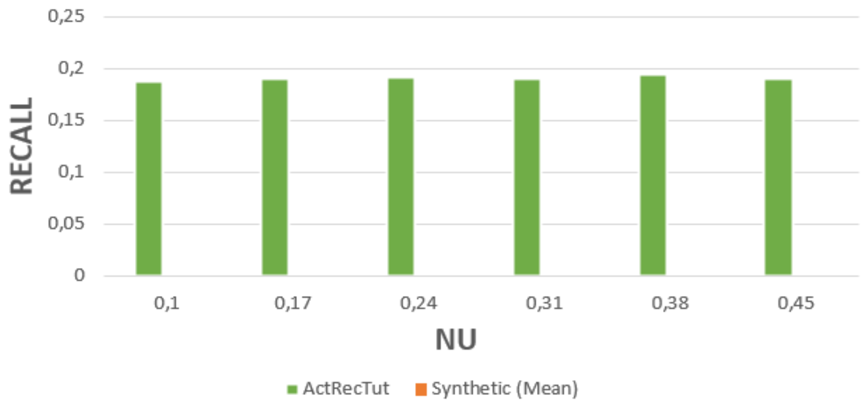


Figure 6.7: Resulting recall obtained by applying One-Class SVM to ActRecTut and Synthetic datasets, fixing contamination = 0.14. For the latter the result was a blank mean and standard deviation of 0.

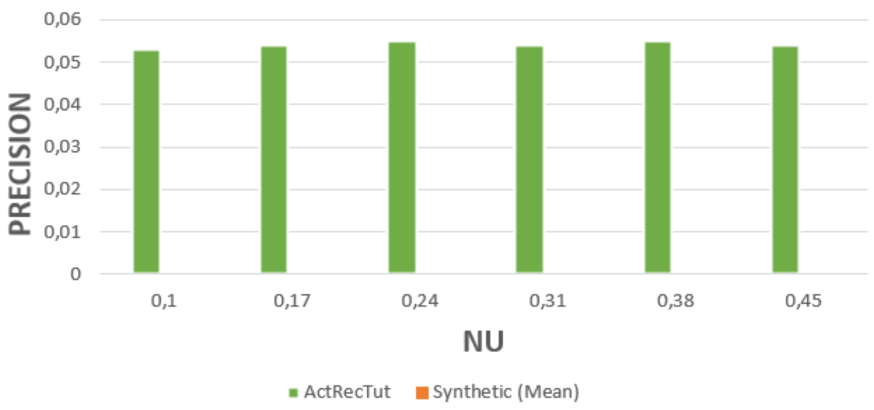


Figure 6.8: Resulting precision obtained by applying One-Class SVM to ActRecTut and Synthetic datasets, fixing contamination = 0.14. For the latter the result was a blank mean and standard deviation of 0.

Therefore, we can conclude that even if our best performer, the LOF algorithm, seems to achieve a seemingly satisfying recall, which is our most important metric, it doesn't provide consistent results, with a low precision performance.
So, like Time2State, these algorithms also don't appear to be the solution for the task at hand.

6.2 Supervised

The supervised analysis is very interesting as it begins to demonstrate some of the improvements we were able to achieve in this work.
Since the hyperparameters used for the three algorithms are the same, but the choices for the number of Max_Features change depending on the dataset (this is because ActRecTut has many more channels than the synthetic datasets), we have grouped the plots according to the datasets.
In Fig.(6.9) and Fig.(6.10), the three algorithms applied to ActRecTut are compared. In particular, the Random Forest does not seem to perform consistently, achieving perfect precision for Max_Features equal to [5,7], but having a poor recall.
On the other hand, our new approach performs consistently well, considering that it consists of simple trees, and moreover its computational time is the lowest among the three.
Even Gradient Boosting achieves reasonably good precision performance but still lacks consistency in recall.

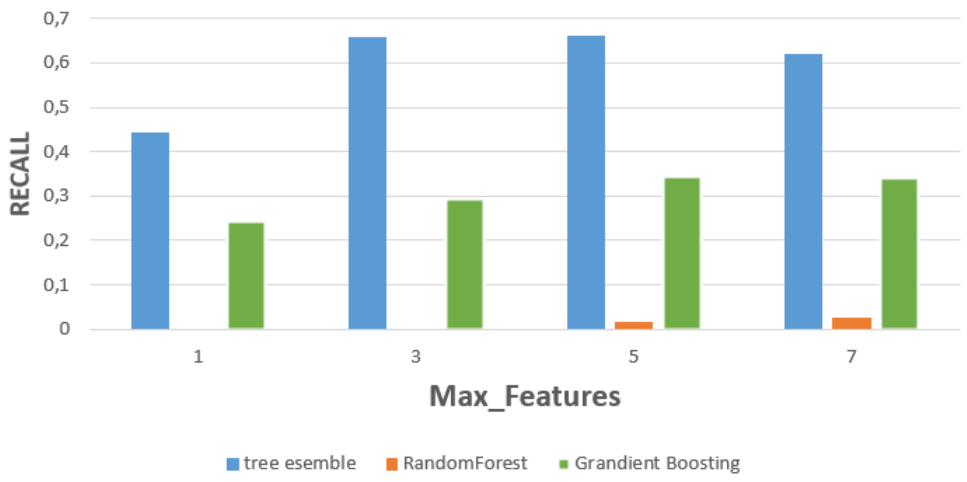


Figure 6.9: Different values of recall obtained by applying Random Forest, Gradient Boosting and our new approach onto ActRecTut. For all the methods we have fixed Max_depth = 7.

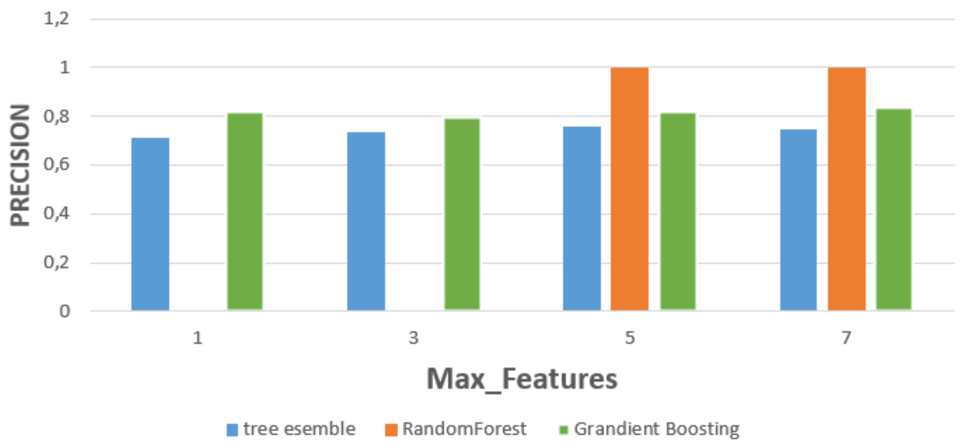


Figure 6.10: Different values of precision obtained by applying Random Forest, Gradient Boosting and our new approach onto ActRecTut. For all the methods we have fixed Max_depth = 7.

The situation changes drastically when we switch to the synthetic datasets. In this case, our new approach is the best one both in terms of recall (see Fig.(6.11)), where it achieves nearly perfect results, and precision (see Fig.(6.12)), where it still obtains a solid score. Another positive aspect is that the associated standard deviation is consistently small, indicating that we obtain consistent results across

all the individual datasets.
The other two methods, particularly Random Forest, perform significantly worse, especially in terms of recall. Moreover, both of them have a consistent standard deviation associated, indicating that they are somewhat sensitive to the data and lack stability.

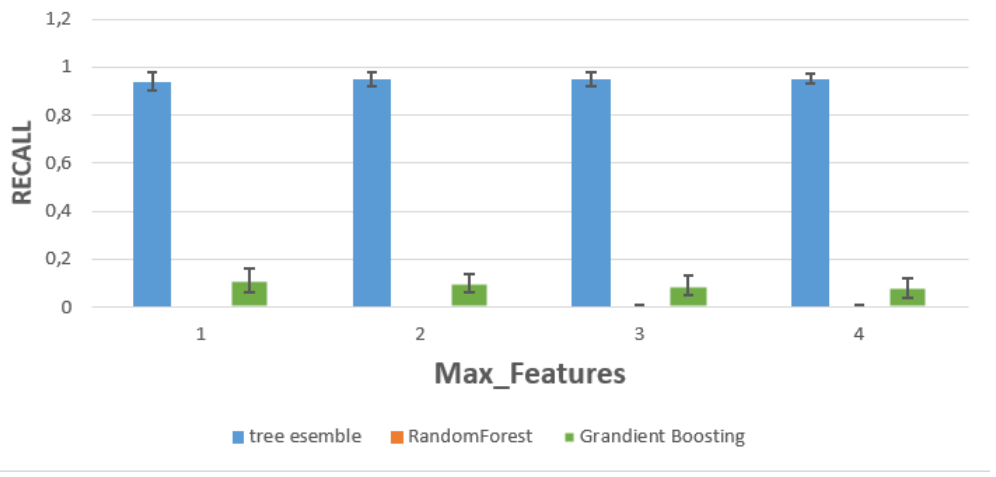


Figure 6.11: Different values of recall obtained by applying Random Forest, Gradient Boosting and our new approach onto the synthetic datasets. For all the methods we have fixed Max_depth = 7. The results shown are the mean ones. The segment present at the top of the bins is the standard deviation associated.

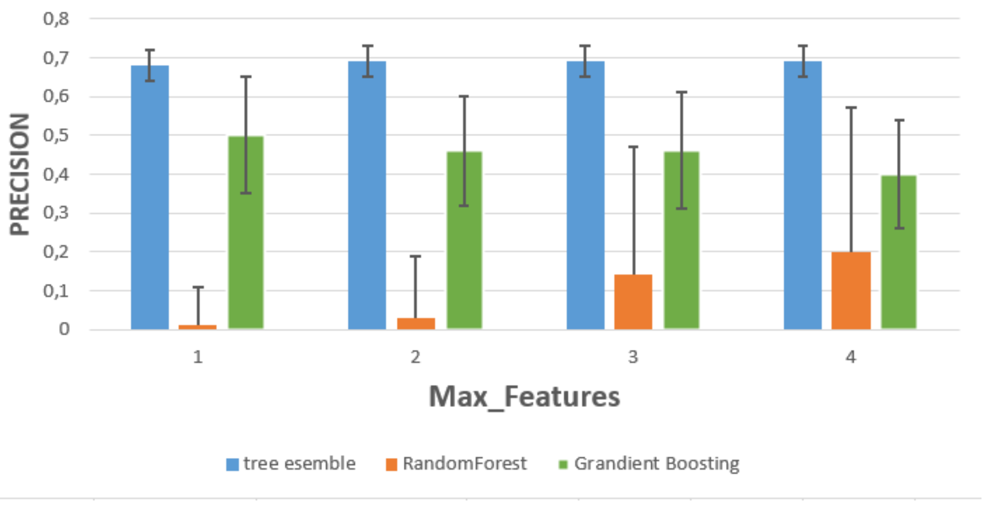


Figure 6.12: Different values of precision obtained by applying Random Forest, Gradient Boosting and our new approach onto the synthetic datasets. For all the methods we have fixed Max_depth = 7. The results shown are the mean ones. The segment present at the top of the bins is the standard deviation associated.

Therefore, even if our testing set isn't complete or wide enough to be certain of it, we can say that from this experiment our approach seems to achieve extremely good results, and moreover, it doesn't suffer from the choice of the dataset, meaning the number of state transitions happening.

6.3 Semi-supervised approach

In this section, we will not only present the raw results obtained with our new pipeline but also the valuable information it provides.
In the previous sections, we didn't delve deeply into this aspect, but it's one of the strengths of our approach. Since we do not train LOF on all the test points but only on the top K points with respect to the b_score, we have access to real-time results.
While we didn't expect to identify all the transitional points among these top K samples, finding the core points or their neighbors would still provide valuable information. After predicting a point in the top K as transitional, we can extend this labeling to the entire test set, marking not only the single point found but also its neighbors as transitional.
For instance, we chose to apply this transformation only to the five points before and after the one found, but with deeper domain knowledge, this parameter could be better defined and selected.
We decided not to apply the same approach to the shapelets case, as its computational time is higher due to the computation of the shapelets themselves, and hence it would result as prohibitive.

6.3.1 Point-wise granularity

For what regards the point-wise case, in Figure (6.13) and Figure (6.14) are shown the basic results obtained on the top K samples of the Synthetic datasets.

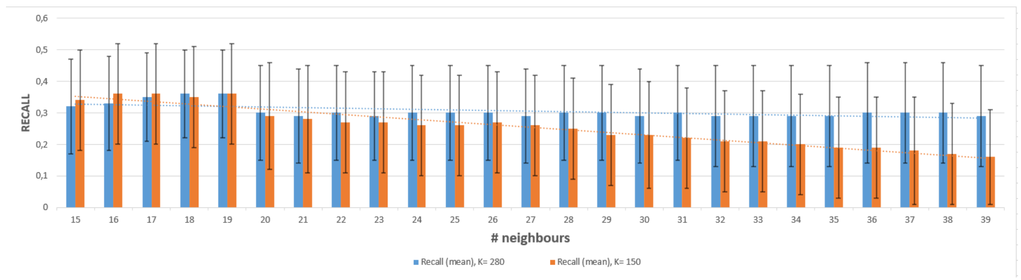


Figure 6.13: The resulting recall obtained by implementing the semi-supervised pipeline on the synthetic datasets is depicted. For this plot we have fixed contamination = 0.19. The presented values represent the mean across all the datasets for each experiment and their associated standard deviation. As the required number of neighbors to classify a point as "normal" increases, we observe a decreasing trend in the corresponding recall. Furthermore, the inclusion of additional samples in the top K appears to enhance the overall results.

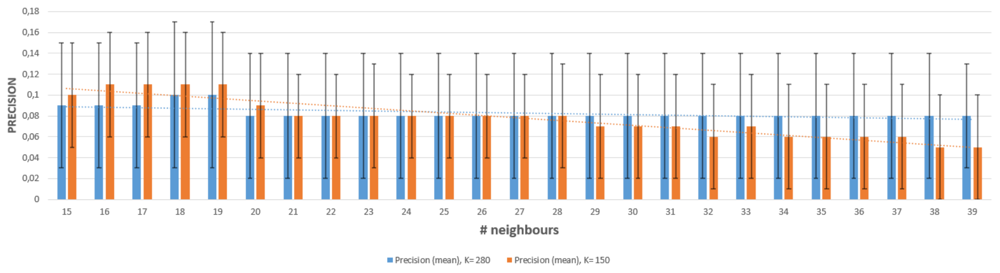


Figure 6.14: The resulting precision obtained by implementing the semi-supervised pipeline on the synthetic datasets is illustrated. For this plot we have fixed contamination = 0.19. The values presented represent the mean across all the datasets for each experiment and their associated standard deviation. Interestingly, the trade-off between the number of K samples considered and performance does not seem to apply to precision, as a lower number of neighbors (K=150) yields better results.

These do not seem to show significant improvements compared to those obtained by the mere application of LOF on the entire series, except for the computational time, which is nearly real-time.
However, as we anticipated in the introduction, the most useful characteristic of this pipeline is the ability to gain insights and information not only about the few transitional points found in the top K samples but also about all the points in the test set. In fact, Figure (6.15) and Figure (6.16) illustrate this feature of the pipeline.

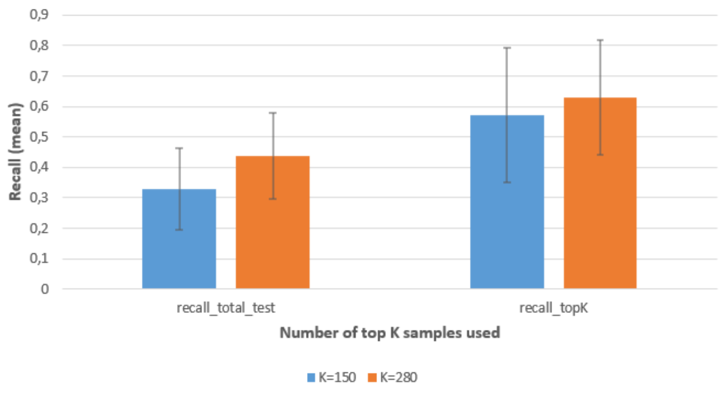


Figure 6.15: New values for the recall of the semi-supervised pipeline on the synthetic datasets. We are still fixing contamination = 0.19 and showing only the case of #neighbours = 19. The measure is computed after incorporating the new information into the predicted labels for the test set. "Recall_total_test" represents the recall obtained by considering the entire test set, whereas "Recall_topK" includes only the top K samples. The results represent the mean of all the experiments conducted on the synthetic datasets and are provided with their corresponding standard deviation.

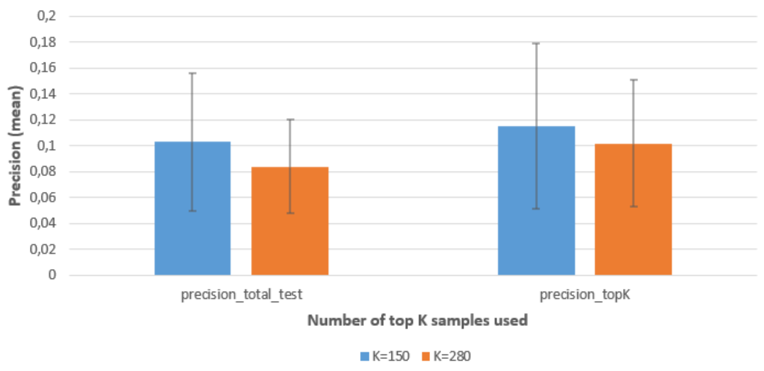


Figure 6.16: New values for the recall of the semi-supervised pipeline on the synthetic datasets. We are still fixing contamination = 0.19 and showing only the case of #neighbours = 19. The measure is computed after incorporating the new information into the predicted labels for the test set. "Precision_total_test" represents the recall obtained by considering the entire test set, whereas "Precision_topK" includes only the top K samples. The results represent the mean of all the experiments conducted on the synthetic datasets and are provided with their corresponding standard deviation.

We selected the two best configurations from Figure (6.13) and, in the applying LOF, leveraged the information about the transitional points we identified. This allowed us to label not only those points but also their previous and subsequent five points in the test set.
Finally, we want to stress that these results should be considered in the context of the challenging nature of the task and the fact that this approach has not been finely tuned yet, leaving room for further improvement.

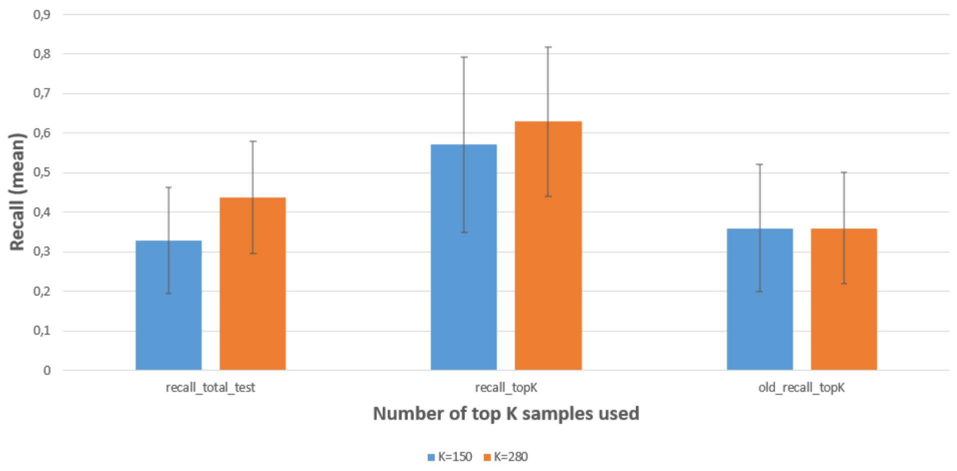
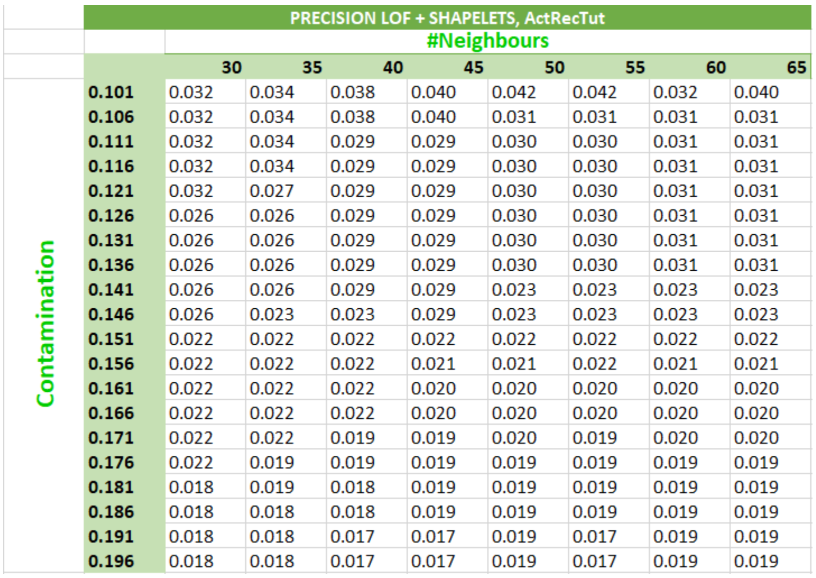


Figure 6.17: Comparison between the recall achieved by the our method before (old_recall_topK) and after the usage of the information gained (recall_total_test and recall_topK). In the plot we have fixed contamination=0.19, #neighbours=19 and shown both K=[150 , 280].

6.3.2 Segment-wise granularity

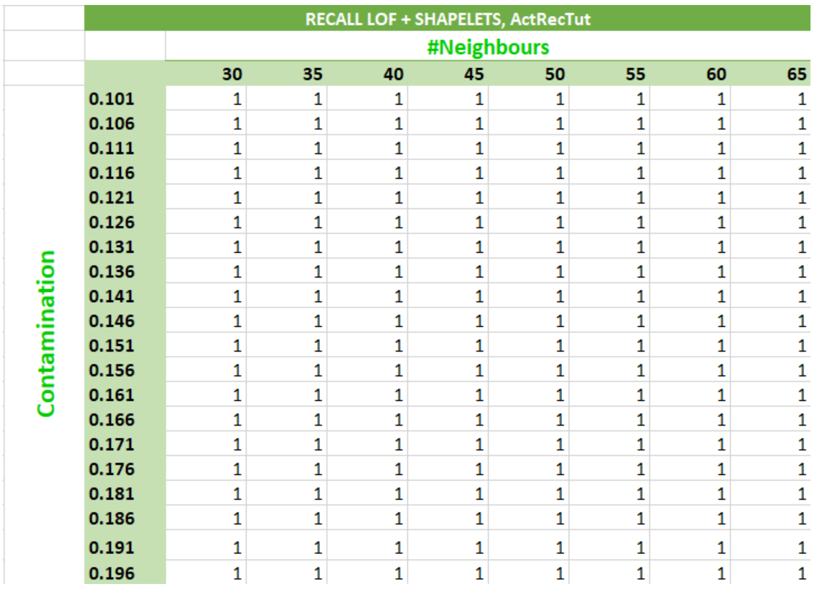
Regarding the experiments on shapelets, we wish to present the results, emphasizing that in this case, we did not conduct as extensive testing as we did for the other methods. To truly assess the most suitable and consistent parameters, a thorough examination of the domain of study, including transitions and their dynamics, should be conducted.
In this work, our primary focus was on assessing how well this pipeline would perform without extensive fine-tuning.
Once again, the results for ActRecTut (Figure (6.18), Figure (6.19)) and the synthetic datasets (Figure (6.20), Figure (6.21)) are significantly different. For the latter, we obtain more consistent results, while for ActRecTut, we observe an optimal recall but near-zero precision.
Drawing definitive conclusions about this pipeline is challenging, and it is clear that more experiments and datasets are needed. In general, both due to the substantial

computational time required and the results obtained, we can conclude that this option does not represent a viable alternative to the currently used methods.



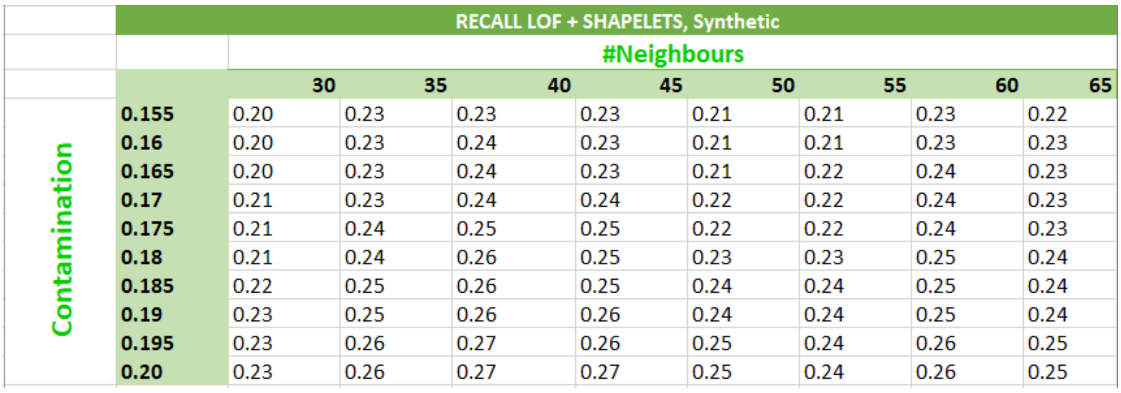
PRECISION LOF + SHAPELETS, ActRecTut								
	#Neighbours							
Contamination	30	35	40	45	50	55	60	65
0.101	0.032	0.034	0.038	0.040	0.042	0.042	0.032	0.040
0.106	0.032	0.034	0.038	0.040	0.031	0.031	0.031	0.031
0.111	0.032	0.034	0.029	0.029	0.030	0.030	0.031	0.031
0.116	0.032	0.034	0.029	0.029	0.030	0.030	0.031	0.031
0.121	0.032	0.027	0.029	0.029	0.030	0.030	0.031	0.031
0.126	0.026	0.026	0.029	0.029	0.030	0.030	0.031	0.031
0.131	0.026	0.026	0.029	0.029	0.030	0.030	0.031	0.031
0.136	0.026	0.026	0.029	0.029	0.030	0.030	0.031	0.031
0.141	0.026	0.026	0.029	0.029	0.023	0.023	0.023	0.023
0.146	0.026	0.023	0.023	0.029	0.023	0.023	0.023	0.023
0.151	0.022	0.022	0.022	0.022	0.022	0.022	0.022	0.022
0.156	0.022	0.022	0.022	0.021	0.021	0.022	0.021	0.021
0.161	0.022	0.022	0.022	0.020	0.020	0.020	0.020	0.020
0.166	0.022	0.022	0.022	0.020	0.020	0.020	0.020	0.020
0.171	0.022	0.022	0.019	0.019	0.020	0.019	0.020	0.020
0.176	0.022	0.019	0.019	0.019	0.019	0.019	0.019	0.019
0.181	0.018	0.019	0.018	0.019	0.019	0.019	0.019	0.019
0.186	0.018	0.018	0.018	0.019	0.019	0.019	0.019	0.019
0.191	0.018	0.018	0.017	0.017	0.019	0.017	0.019	0.019
0.196	0.018	0.018	0.017	0.017	0.019	0.017	0.019	0.019

Figure 6.18: Tabular results for precision obtained with the shapelets unsupervised pipeline on ActRecTut.



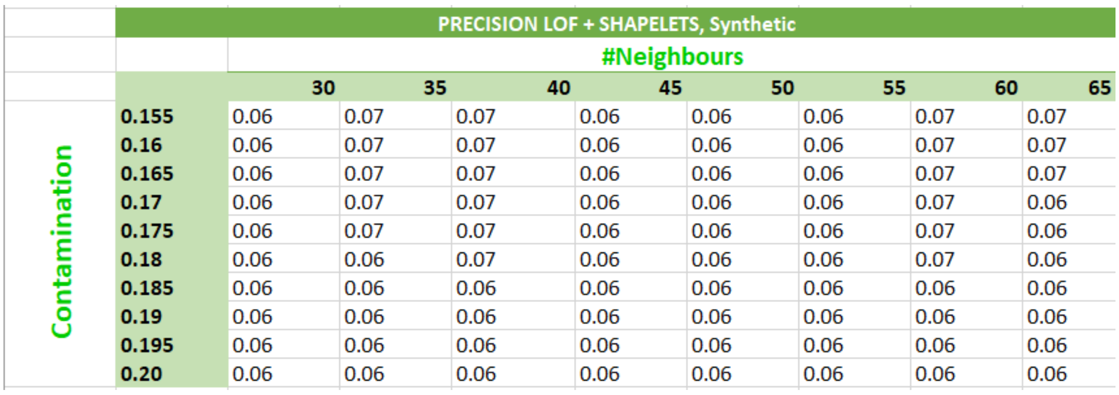
RECALL LOF + SHAPELETS, ActRecTut								
	#Neighbours							
Contamination	30	35	40	45	50	55	60	65
0.101	1	1	1	1	1	1	1	1
0.106	1	1	1	1	1	1	1	1
0.111	1	1	1	1	1	1	1	1
0.116	1	1	1	1	1	1	1	1
0.121	1	1	1	1	1	1	1	1
0.126	1	1	1	1	1	1	1	1
0.131	1	1	1	1	1	1	1	1
0.136	1	1	1	1	1	1	1	1
0.141	1	1	1	1	1	1	1	1
0.146	1	1	1	1	1	1	1	1
0.151	1	1	1	1	1	1	1	1
0.156	1	1	1	1	1	1	1	1
0.161	1	1	1	1	1	1	1	1
0.166	1	1	1	1	1	1	1	1
0.171	1	1	1	1	1	1	1	1
0.176	1	1	1	1	1	1	1	1
0.181	1	1	1	1	1	1	1	1
0.186	1	1	1	1	1	1	1	1
0.191	1	1	1	1	1	1	1	1
0.196	1	1	1	1	1	1	1	1

Figure 6.19: Tabular results for recall obtained with the shapelets unsupervised pipeline on ActRecTut



		RECALL LOF + SHAPELETS, Synthetic							
		#Neighbours							
		30	35	40	45	50	55	60	65
Contamination	0.155	0.20	0.23	0.23	0.23	0.21	0.21	0.23	0.22
	0.16	0.20	0.23	0.24	0.23	0.21	0.21	0.23	0.23
	0.165	0.20	0.23	0.24	0.23	0.21	0.22	0.24	0.23
	0.17	0.21	0.23	0.24	0.24	0.22	0.22	0.24	0.23
	0.175	0.21	0.24	0.25	0.25	0.22	0.22	0.24	0.23
	0.18	0.21	0.24	0.26	0.25	0.23	0.23	0.25	0.24
	0.185	0.22	0.25	0.26	0.25	0.24	0.24	0.25	0.24
	0.19	0.23	0.25	0.26	0.26	0.24	0.24	0.25	0.24
	0.195	0.23	0.26	0.27	0.26	0.25	0.24	0.26	0.25
	0.20	0.23	0.26	0.27	0.27	0.25	0.24	0.26	0.25

Figure 6.20: Tabular results for recall obtained with the shapelets unsupervised pipeline on the synthetic datasets.



		PRECISION LOF + SHAPELETS, Synthetic							
		#Neighbours							
		30	35	40	45	50	55	60	65
Contamination	0.155	0.06	0.07	0.07	0.06	0.06	0.06	0.07	0.07
	0.16	0.06	0.07	0.07	0.06	0.06	0.06	0.07	0.07
	0.165	0.06	0.07	0.07	0.06	0.06	0.06	0.07	0.07
	0.17	0.06	0.07	0.07	0.06	0.06	0.06	0.07	0.06
	0.175	0.06	0.07	0.07	0.06	0.06	0.06	0.07	0.06
	0.18	0.06	0.06	0.07	0.06	0.06	0.06	0.07	0.06
	0.185	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
	0.19	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
	0.195	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06
	0.20	0.06	0.06	0.06	0.06	0.06	0.06	0.06	0.06

Figure 6.21: Tabular results for precision obtained with the shapelets unsupervised pipeline on the synthetic datasets.

Chapter 7

Conclusions and Future's works

7.1 Conclusions

In this work, we present a novel perspective on the problem of time series segmentation. Instead of focusing on the states themselves, we have chosen the transitional points as the target of our analysis. These transitional points represent the samples that stand between a change of state. This task is considerably more challenging, as the new class introduced cannot be considered solely composed of outliers; it actually encompasses those points with features from two different states.
We have tested all of our approaches on ActRecTut, a real-world dataset, as well as on one hundred synthetic datasets.
To address the task at hand, we first implemented a new supervised pipeline based on tree ensembles and a novel sampling technique. This approach has yielded significantly more promising and consistent results than the traditional random forest and gradient boosting methods, with the added benefit of a reduced runtime. Furthermore, to assess the necessity of a new approach for this task, we tested Time2State, a state-of-the-art unsupervised algorithm commonly used in the context of time series segmentation, which can be readily adapted for transitional points recognition.
However, our analysis indicates that this method performs poorly. Therefore, we attempted to enhance its results by creating a more complex pipeline, incorporating the b_score as a performance measure, and using PYOD's outlier detection algorithms as classifiers. This method allowed us to gather valuable information that, when integrated into our pipeline, led to more consistent and generally superior results compared to Time2States, while also reducing the runtime.

7.2 Future's work

Throughout the development of this work, we encountered several challenges in relation to the testing datasets. To enhance our result assessment, it would be advantageous to create a more diverse collection of datasets tailored specifically for this task.
Furthermore, incorporating this approach into a segmentation-focused pipeline could aid in more accurately identifying the state boundaries, resulting in higher overall precision.
Lastly, a deeper understanding of the domain, including insights into transitions, their duration, and behaviors, would undoubtedly improve the grid search for the hyperparameters utilized in our algorithms.

Bibliography

[1] Ardi Tampuu Zurab Bzhalava. «ViraMiner: Deep learning on raw DNA sequences for identifying viral genomes in human samples». In: (2019).

[2] Chengyu Wang Kui Wu Tongqing Zhou Guang Yu Zhiping Cai. «TSAGen: Synthetic Time Series Generation for KPI Anomaly Detection». In: (2022).

[3] Chengyu Wang Kui Wu Tongqing Zhou Zhiping Cai. «Time2State: An Unsupervised Framework for Inferring the Latent States in Time Series Data». In: (2023).

[4] Patricia Garcia-Canadilla Sergio Sánchez Martínez Fatima Crispi. «Machine Learning in Fetal Cardiology: What to Expect». In: (2020).

[5] Aminikhanghahi S Cook DJ. «A survey of methods for time series change point detection». In: (2017).

[6] Johann Faouzi. «Time Series Classification: A review of Algorithms and Implementations». In: (2022).

[7] Yifan Sun Yaqi Duan Hao Gong and Mengdi Wang. «Learning low dimensional state embeddings and metastable clusters from time series». In: (2019).

[8] C. Grunau and V. Rozhon. «Adapting k-means algorithms for outliers». In: (2020).

[9] *https://developer.ibm.com/learningpaths/get-started-time-series-classification-api/what-is-time-series-classification.*

[10] *https://en.wikipedia.org/wiki/Artificial*$_n$*eural*$_n$*etwork.*

[11] *https://en.wikipedia.org/wiki/Association*$_r$*ule*$_l$*earning.*

[12] *https://en.wikipedia.org/wiki/Confusion*$_m$*atrix.*

[13] *https://en.wikipedia.org/wiki/DBSCAN.*

[14] *https://en.wikipedia.org/wiki/Decision*$_t$*ree.*

[15] *https://en.wikipedia.org/wiki/Gradient*$_b$*oosting.*

[16] *https://en.wikipedia.org/wiki/Isolation*$_f$*orest.*

[17] *https://en.wikipedia.org/wiki/K-means$_c$lustering.*

[18] *https://en.wikipedia.org/wiki/Local$_o$utlier$_f$actor.*

[19] *https://en.wikipedia.org/wiki/Logistic$_r$egression.*

[20] *https://en.wikipedia.org/wiki/Loss$_f$unction.*

[21] *https://en.wikipedia.org/wiki/Natural$_l$anguage$_p$rocessing.*

[22] *https://en.wikipedia.org/wiki/Precision$_a$nd$_r$ecall.*

[23] *https://en.wikipedia.org/wiki/Random$_f$ores.*

[24] *https://en.wikipedia.org/wiki/Silhouette$_($clustering).*

[25] *https://en.wikipedia.org/wiki/Spectral$_c$lustering.*

[26] *https://en.wikipedia.org/wiki/Support$_v$ector$_m$achine.*

[27] *https://en.wikipedia.org/wiki/Time$_s$eries.*

[28] *https://en.wikipedia.org/wiki/Unsupervised$_l$earning.*

[29] *https://en.wikipedia.org/wiki/Weak$_s$upervision.*

[30] *https://github.com/andreas-bulling/ActRecTut.*

[31] *https://medium.com/@mail.garima7/one-class-svm-oc-svm-9ade87da6b10.*

[32] *https://pyod.readthedocs.io/en/latest/index.html.*

[33] *https://tslearn.readthedocs.io/en/stable/user$_g$uide/shapelets.html.*

[34] *https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/.*

[35] *https://www.cs.ucr.edu/ eamonn/time$_s$eries$_d$ata/.*

[36] *https://www.datanovia.com/en/lessons/cluster-validation-statistics-must-know-methods/.*

[37] *https://www.geeksforgeeks.org/apriori-algorithm/.*

[38] *https://www.geeksforgeeks.org/clustering-in-machine-learning/.*

[39] *https://www.ibm.com/topics/machine-learning.*

[40] *https://www.ibm.com/topics/supervised-learning.*

[41] *https://www.v7labs.com/blog/semi-supervised-learning-guide.*

[42] Jean-Yves Franceschi Aymeric Dieuleveut Martin Jaggi. «Unsupervised Scalable Representation Learning for Multivariate Time Series». In: (2020).

[43] L. Ye E. Keogh. «Time series shapelets: a new primitive for data mining». In: (2009).

[44] Yue Lu Renjie Wu Abdullah Mueen Maria A. Zuluaga Eamonn Keogh. «Matrix Profile XXIV: Scaling Time Series Anomaly Detection to Trillions of Datapoints and Ultra-fast Arriving Data Streams». In: (2022).

[45] Arik Ermshaus Patrick Schafer Ulf Leser. «ClaSP - Parameter-free Time Series Segmentation». In: (2022).

[46] L. Shen Z. Li and J. Kwok. «Timeseries anomaly detection using temporal hierarchical one-class network». In: (2020).

[47] Srikanth Thudumu Philip Branch Jiong Jin Jugdutt (Jack) Singh. «A comprehensive survey of anomaly detection techniques for high dimensional big data». In: (2020).

[48] Charles Truonga Laurent Oudreb Nicolas Vayatisa. «Selective review of offline change point detection methods». In: (2020).

[49] Y. Zhao X. Hu C. Cheng C. Wang C. Wan W. Wang J. Yang H. Bai Z. Li C. Xiao. «Accelerating large-scale unsupervised heterogeneous outlier detection». In: (2021).

[50] W. Yu J. Li M. Z. A. Bhuiyan R. Zhang and J. Huai. «Ring: Real-time emerging anomaly monitoring system over text streams». In: (2017).

[51] S. Han X. Hu H.Huang M. Jiang Y. Zhao. «Anomaly Detection Benchmark». In: (2022).