Learning models on healthcare data with quality indicators

Donato Tiano

THESE de DOCTORAT DE
L'UNIVERSITE CLAUDE BERNARD LYON 1

Ecole Doctorale N° 512
InfoMaths

Discipline : Informatique

Soutenue publiquement le 08/12/2022, par :
Donato Tiano

Learning Models on Healthcare Data with Quality Indicators

Modèles d'apprentissage sur les données de santé avec indicateurs de qualité

Devant le jury composé de :

AUSSEM Alexandre Professeur des Universités, Université Lyon 1	**Examinateur**
BOUCELMA Omar Professeur des Universités, Aix-Marseille Université	**Rapporteur**
ZEITOUNI Karine Professeure des Universités, Université Paris-Saclay	**Rapporteure**
BEN MOKHTAR Sonia Directrice de Recherche, CNRS Lyon	**Présidente**
DUMBRAVA Stefania Gabriela Maître de Conférences, ENSIIE Paris	**Examinatrice**
BONIFATI Angela Professeure des Universités, Université Lyon 1	**Directrice de thèse**
BIFET Albert Professeure des Universités, Université de Waikato	**Invité**

Resumé en Français

L'évolution incessante de la technologie nous a permis de recueillir des données sur chaque type de phénomène. Cette amélioration a été rendue possible par la multiplication des sources à partir desquelles de nouvelles données peuvent être collectées. Les données collectées par ces sources sont très diverses, mais les données les plus couramment recueillies sont les séries temporelles. Les séries temporelles sont des collections de données obtenues par des mesures répétées dans le temps. Elles sont généralement représentées par l'axe cartésien, où l'axe des y représente les valeurs détectées, et l'axe des x le temps de détection. La figure 1.1 est un exemple de série temporelle. Cette collecte de données vise à fournir des éléments de réflexion pour l'extraction d'événements dans les données. L'extraction de données vise à extraire des informations des ensembles de données et à les représenter dans une configuration compréhensible pour une utilisation ultérieure, telle que la détection d'anomalies, le regroupement, l'analyse prédictive, etc. L'ensemble du processus de découverte et d'extraction de modèles à partir de l'ensemble de données s'effectue par le biais de plusieurs techniques d'extraction, notamment l'apprentissage automatique, les statistiques et les systèmes de base de données.

Les séries temporelles sont probablement le principal type de données collectées et ensuite évaluées par les systèmes d'exploration de données. Ce type de données peut représenter des données collectées dans différents domaines, tels que la surveillance de la santé, les appareils ménagers et les machines industrielles. Par conséquent, l'extraction de séries temporelles (TSM) est probablement le domaine le plus étudié de tous les sujets d'extraction de données.

Ce domaine est ensuite divisé par le nombre de sources adoptées pour surveiller un phénomène. Par exemple, dans le cadre d'une simple surveillance cardiaque, les données sont généralement collectées par une seule source (ECG), mais cela ne suffit souvent pas pour découvrir certaines pathologies. D'autres sources doivent donc être adoptées pour obtenir des informations plus précises. Cette différenciation conduit à deux types de séries temporelles : Les séries temporelles univariées (UTS) lorsque la source de données est unique, et les séries temporelles multivariées (MTS) lorsque la source de données est multiple.

La série chronologique n'est pas une structure simple. Chaque observation de la série a une relation forte avec les autres observations. Cette interrelation est la caractéristique principale des séries temporelles, et toute opération d'extraction de séries temporelles doit y faire face. L'interrelation est souvent révélée sur l'axe temporel, par exemple, les anciennes observations peuvent affecter les récentes, ou sur les signaux, par exemple, une source de données peut interagir avec les autres pour générer les données à chaque horodatage. La solution adoptée pour gérer l'interrelation est liée aux opérations d'extraction. Par exemple,

la recherche de certaines observations cycliques est une opération fréquente pour découvrir des anomalies. En effet, dans le cas d'anomalies, ces cycles peuvent être interrompus. Alors que dans le regroupement de séries temporelles, les algorithmes négligent généralement des sous-ensembles d'observations communes dans certaines séries pour les regrouper.

Le principal problème de la plupart de ces techniques d'exploration est de ne pas adopter d'opération de prétraitement sur les séries temporelles. En effet, de nombreux algorithmes utilisent les séries temporelles brutes pour en extraire des informations utiles à l'utilisateur. Cependant, les séries chronologiques brutes présentent de nombreux effets indésirables, tels que des points noisy ou l'énorme espace mémoire requis pour les longues séries. Dans cette thèse, nous proposons de nouvelles techniques d'exploration de données basées sur l'adoption des caractéristiques les plus représentatives des séries temporelles pour obtenir de nouveaux modèles à partir des données. L'adoption de ces caractéristiques a un impact considérable sur l'évolutivité des systèmes. En effet, l'extraction d'une caractéristique de la série temporelle permet de réduire une série entière en une seule valeur. Par conséquent, cela permet d'améliorer la gestion des séries temporelles, en réduisant la complexité des solutions en termes de temps et d'espace.

L'évolutivité n'est pas le seul avantage de l'adoption des caractéristiques des séries temporelles. La plupart des solutions ont récemment adopté des techniques d'extraction de caractéristiques basées sur l'analyse en composantes principales (ACP) ou les réseaux neuronaux. Les caractéristiques obtenues à partir de ces processus sont incompréhensibles pour l'homme, ce qui signifie qu'elles ne permettent pas une étude plus approfondie. Dans cette thèse, nous adoptons des caractéristiques interprétables qui permettent une étude plus approfondie lorsque nos algorithmes obtiennent de nouvelles solutions. De plus, nous proposons une nouvelle technique de sélection des caractéristiques pour réduire le nombre de caractéristiques extraites et l'interprétabilité des résultats.

La figure 1.2 montre les principaux problèmes rencontrés au cours de cette thèse. Nous présentons les solutions de l'état de l'art pour deux sujets, puis nous présentons nos solutions. Les problèmes étudiés sont divisés en deux sujets principaux : (i) Dans le premier, nous avons étudié le clustering pour les données de séries temporelles univariées et multivariées. Dans la solution univariée, nous proposons un algorithme pour détecter la relation globale entre les séries en exploitant les caractéristiques statistiques. Dans la solution multivariée, nous proposons un algorithme qui exploite à la fois les caractéristiques intra-signal, caractérisant les signaux uniques des séries temporelles multivariées, et les caractéristiques inter-signal mesurant la relation par paire (en termes de similarité et de corrélation) des signaux multiples en utilisant des métriques interprétables. (ii) Dans le deuxième thème, nous avons exploré les algorithmes de détection des anomalies dans le contexte streaming. Nous proposons ensuite

un algorithme de détection des anomalies qui exploite les caractéristiques statistiques pour détecter les changements dans les données en streaming. De plus, l'algorithme combine le clustering et l'algorithme de réseau neuronal pour améliorer sa qualité.

Le temps est une dimension qui affecte de nombreux aspects des phénomènes du monde réel et du monde numérique. L'environnement physique, les machines industrielles, le suivi des soins de santé et les activités économiques et financières sont quelques exemples d'applications dont les composants sont régulés et évoluent dans le temps. Le regroupement de séries temporelles est un problème récurrent dans les applications réelles impliquant des pipelines de science des données et d'analyse des données. Il vise à organiser des objets de données non étiquetés en groupes homogènes tout en minimisant la dissimilarité intra-classe et en maximisant la dissimilarité inter-classe. Les systèmes existants s'appuient souvent sur des mesures de distance entre des séries temporelles brutes et ne parviennent pas à garantir l'interprétabilité des résultats tout en préservant l'efficacité et l'évolutivité. Cela entrave l'intervention humaine et son droit à l'explicabilité, en particulier pour les experts du domaine qui ne sont pas familiers avec le le Machine Learning.

Dans la solution FeatTS, nous proposons une méthode de clustering pour les séries temporelles univariées qui extrait les caractéristiques les plus représentatives de la série. Les séries temporelles univariées sont des séries temporelles où chaque phénomène est mesuré par une seule source. Pour ce problème, la plupart des solutions adoptent la forme de la série pour découvrir des similarités dans les données. Cette méthode est puissante dans les séries temporelles très particulières. De plus, dans le cas de très longues séries, la recherche de particularités nécessite beaucoup de temps et de mémoire. Notre solution vise à adopter les particularités en les convertissant en réseaux de graphes pour extraire les interrelations entre les signaux en adoptant des algorithmes de détection de communautés. Ensuite, une matrice de cooccurrence fusionne toutes les communautés détectées. L'intuition est que si deux séries temporelles sont similaires, elles appartiennent souvent à la même communauté, et la matrice de cooccurrence permet de le révéler. Nous résumons nos principales contributions comme suit :

- Nous présentons une nouvelle méthode de Semi Supervised Clustering qui tire parti des caractéristiques les plus discriminantes extraites des séries chronologiques.
- Notre méthode permet de traiter à égalité toutes les caractéristiques d'un ensemble de données donné au lieu de présélectionner un nombre fixe d'entre elles pour tous les ensembles de données ou de se contenter de tirer parti de la similarité des données brutes.

- Notre méthode permet d'obtenir des résultats de plus haute qualité que les dernières lignes de base sur les ensembles de données de la littérature.

Dans Time2Feat, nous créons un nouveau clustering de séries temporelles multivariées. Dans les séries temporelles multivariées, la mesure du phénomène étudié est détectée à travers différentes sources. En raison de leurs besoins élevés en mémoire, la plupart des solutions proposées dans la littérature visent à réduire la taille des séries en appliquant des algorithmes d'extraction de caractéristiques tels que l'ACP ou le réseau neuronal. Ces solutions sont très gourmandes en temps et ne fournissent souvent aucune caractéristique discriminante. C'est pourquoi Time2Feat propose deux extractions différentes pour améliorer la qualité des caractéristiques. Le premier type d'extraction est appelé extraction de caractéristiques intra-signal et permet d'obtenir des caractéristiques à partir de chaque signal de la série temporelle multivariée. L'autre type d'extraction est appelé Inter-Signal Features Extraction et permet d'obtenir des caractéristiques en considérant des couples de signaux appartenant à la même série temporelle multivariée. En outre, les deux méthodes fournissent des caractéristiques interprétables, ce qui rend possible une analyse ultérieure. Grâce à l'adoption des caractéristiques, l'ensemble du processus de clustering des séries temporelles est plus léger, ce qui réduit le temps nécessaire pour obtenir le cluster final. Par conséquent, les deux solutions représentent l'état de l'art dans leur domaine. Les principales contributions sont résumées comme suit :

- Un système de regroupement de bout en bout interprétable et efficace pour les séries temporelles multivariées.
- Un système de regroupement humain dans la boucle permettant des annotations basées sur l'apprentissage.
- Une évaluation complète et une source ouverte d'artefacts.

La recherche de sous-séquences anormales dans les données de séries temporelles est un problème crucial dans divers contextes, des applications financières au suivi des soins de santé. Une anomalie peut signifier des événements critiques, qu'il est essentiel de révéler au plus vite. Ces dernières années, de nombreux algorithmes ont été proposés pour détecter ces anomalies, et la plupart d'entre eux sont basés sur deux techniques principales : Le clustering et les réseaux neuronaux. Le clustering est adopté pour découvrir si les nouveaux points obtenus sont placés à un endroit connu dans la dimension spatiale, et la plupart du temps, il est adopté pour les séries temporelles hors ligne. En revanche, les réseaux neuronaux sont généralement adoptés pour les séries chronologiques en ligne, et ils prévoient les points futurs

qui seront comparés à l'original. Les points originaux sont considérés comme anormaux si la distance entre les points est élevée.

Dans AnomalyFeat, nous proposons un algorithme pour révéler des anomalies à partir de séries temporelles univariées. La caractéristique de cet algorithme est la capacité de travailler parmi des séries temporelles en ligne, c'est-à-dire que chaque valeur de la série est obtenue en streaming. Dans la continuité des solutions précédentes, nous adoptons les fonctionnalités de révélation des anomalies dans les séries. Avec AnomalyFeat, nous unifions les deux algorithmes les plus populaires pour la détection des anomalies : le clustering et le réseau neuronal récurrent. Nous cherchons à découvrir la zone de densité du nouveau point obtenu avec le clustering. En revanche, nous adoptons le réseau neuronal récurrent pour prédire le nouveau point, pour savoir s'il sera similaire au point réel et si les deux ont été placés dans le même cluster.

Abstract

Time Series is a set of observations arranged chronologically. Time Series usually result from observing some phenomenon analyzed in the real world, such as medical monitoring, financial analysis, and industrial machinery. Many of these applications are addressed in the literature by various tasks such as Classification, Anomaly Detection, Prediction, and Clustering. In this thesis, we propose new methods for facing two of these tasks tasks, namely Clustering and Anomaly Detection, by adopting the characteristics of the series. The main advantage of adopting the characteristics is their natural explainability. Indeed, each feature is obtained through a series of statistical features, in most cases understandable by humans. The adoption of these features guarantees the interpretability of the results while at the same time preserving efficiency and scalability.

For what concerns the clustering task, we propose FeatTS and Time2Feat, respectively for univariate and multivariate time series. FeatTS is a feature-based semi supervised clustering framework addressing the above issues for variable length and heterogeneous time series. Specifically, FeatTS leverages a graph encoding of the time series obtained by considering a high number of extracted features. It then employs community detection and builds upon a Co-Occurrence matrix to unify all the best clustering results. FeatTS outperforms the state-of-the-art clustering methods and is the first to digest domain-specific time series such as healthcare time series while still being robust and scalable.

Time2Feat is an end-to-end machine learning system for multivariate time series (MTS) clustering. The system is the first to leverage the time series' inter-signal and intra-signal features. While relying on state-of-the-art feature extraction approaches allows further refine the features by choosing the most appropriate ones and incorporating human feedback in the feature selection process. Time2Feat also enables users to inspect MTS further, compare the state-of-the-art methods, analyze the features, and cluster results. The system leverages human knowledge to improve the clustering pipeline using human annotations. We demonstrate the effectiveness, interpretability, efficiency, and robustness of Time2Feat through experiments on eighteen real-world and benchmarking time series datasets, comparing them with state-of-the-art MTS clustering methods.

For what concerns anomaly detection,, we propose AnomalyFeat, a new method that leverages the features for recognizing anomalous points in streaming data by learning incremental Neural Networks and creating clusters of points with similar characteristics. This method is innovative as it combines unsupervised learning with supervised learning for capturing the anomalies of time series in an incremental fashion. Preliminary experiments show the promising effectiveness and efficiency of the approach.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

The relentless evolution of technology has allowed us to gather data from each kind of phenomenon. This improvement has been made possible by increasing sources from which new data can be collected. The data collected by these sources are very diverse, but the most commonly gathered data are temporal series. The temporal series are collections of data obtained through repeated measurements over time. They are usually represented through the Cartesian Axis, where the y-axis represents the values detected, and the x-axis is the detection time.

Figure 1.1 is an example of temporal series. This data collection aims to provide some food for thought for mining events inside the data. Data Mining aims to extract information from the datasets and represent them into an understandable configuration for further use, such as anomaly detection, clustering, predictive analysis, et cetera. The entire process of discovering and extracting patterns from the dataset takes place through several mining techniques, including machine learning, statistics, and database systems.

The temporal series is probably the primary data type collected and then evaluated by the data mining systems. This kind of data can represent data collected from different domains, such as health monitoring, household appliances, and industrial machines. Therefore, Time Series Mining(TSM) is presumably the vastest studied area of all the Data Mining topics.

This area is subsequently split by the number of sources adopted for monitoring a phenomenon. For example, in simple heart monitoring, the data are usually collected by only one source (ECG) but often is not enough to discover some pathologies. Hence, other sources need to be adopted to obtain more precise information. This differentiation leads to two kinds of time series: Univariate Time Series(UTS) when the data source is unique, and Multivariate Time Series(MTS) when the data source is more than one.

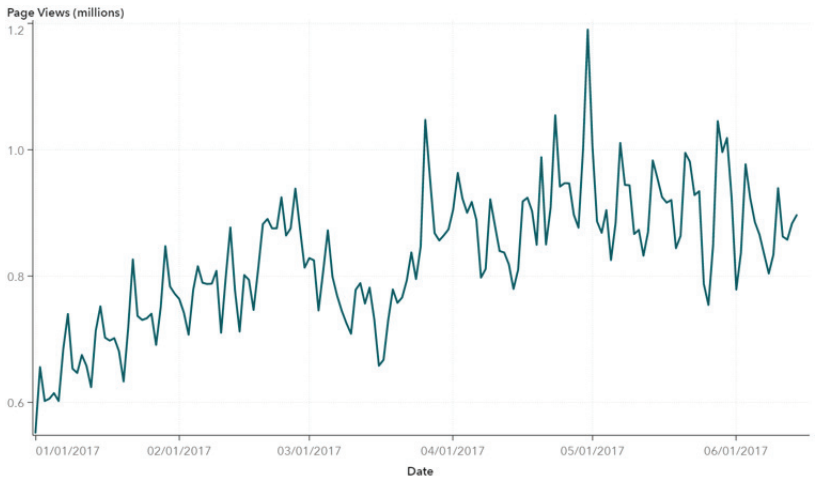


Fig. 1.1 Example of Time Series

The time series is not an easy structure. Each observation in the series has a strong relationship with the other observations. This inter-relationship is the primary characteristic of the time series, and any time series mining operation has to deal with it. The inter-relationship is often revealed on the temporal axis, e.g., the old observations may affect the recent ones, or on the signals, e.g., one data source may interact with the others to generate the data at each timestamp. The solution adopted for managing the inter-relationship is related to the mining operations. For instance, looking for some cyclic observations is a frequent operation for discovering anomalies. Indeed, in the case of anomalies, these cycles may be interrupted. While in the clustering of time series, the algorithms usually overlook subsets of observations in common in some series to group them.

The majority of mining techniques employed suffer from a crucial deficiency, namely the absence of any preprocessing operations performed on the time series data. Indeed, many algorithms threaten raw time series for extracting helpful information for the user. Nevertheless, raw time series carries numerous adverse effects, such as noisy points or the huge memory space required for long series. In this thesis, we propose new data mining techniques based on adopting the most representative characteristics of the time series for obtaining new patterns from the data. Adopting the features has a profound impact on the scalability of the systems. Indeed, extracting a characteristic from the time series allows the reduction of an entire series into one value. Therefore, it permits improving time series management, reducing the complexity of solutions in terms of time and space for each length of time series.

In addition to scalability, adopting the characteristics of time series data offers other advantages as well. While many solutions use feature extraction techniques such as Principal

Component Analysis (PCA) or Neural Networks, the resulting features are often unintelligible to humans, limiting further analysis. In this thesis, we propose the use of interpretable features that allow for further investigation when new solutions are obtained using our algorithms. Furthermore, we introduce a novel feature selection technique to reduce the number of extracted features and increase the interpretability of the results.

My thesis has been funded by ANR under the contract ANR QualiHealth[1].

1.2 Contributions

Figure 1.2 shows the main tasks faced during this thesis. We present state-of-the-art solutions for two topics, and then we present our solutions. The problems studied are divided into two main topics: (i) In the first, we studied the clustering for the univariate and multivariate time series data. In the univariate solution, we propose an algorithm for detecting the global relationship between the series by leveraging the statistical features. In the multivariate solution, we propose an algorithm that exploits both intra-signal features, characterizing the single signals of multivariate time series, and inter-signal features measuring pairwise relatedness (in terms of similarity and correlation) of multiple signals employing interpretable metrics. (ii) In the second topic, we explored algorithms for detecting anomalies in the streaming context. Then, we propose an anomaly detection algorithm that leverages the statistical features for detecting changement in the streaming data. Moreover, the algorithm combines the clustering and the neural network algorithm to improve its quality.

1.2.1 Time Series Clustering

Time series clustering is a recurrent problem in real-life applications involving data science and data analytics pipelines. It aims to organize unlabeled data objects into homogeneous groups while minimizing intra-cluster dissimilarity and maximizing inter-cluster dissimilarity. Existing systems often leverage distance measures between raw time series and fail to guarantee the interpretability of the results while at the same time preserving efficiency and scalability. This hinders human intervention and their right to explainability, especially for domain experts unfamiliar with ML.

1.2.1.1 Limitation of Current Approaches

The main limitation in univariate time series clustering is the series size. Indeed, many approaches adopt the distance between the series to evaluate the similarities, and in very long

[1] https://anr.fr/Project-ANR-18-CE23-0002

series, the procedure can require much time. Other methods have been proposed to adopt the features for clustering the time series[142, 111]. However, many apply the same features for every kind of dataset without any feature selection algorithm for detecting the most characteristic features. Research on MTS is still at an early stage. Proposals adapt clustering approaches designed for univariate to multivariate time series after applying dimensionality reduction techniques. Examples of such techniques (CSPCA[76] and MC2PCA[75]) are based on the Principal Component Analysis (PCA), which enables the conversion of correlated features in the high dimensional space into a set of uncorrelated features in the low dimensional space. Nevertheless, the resulting clusters suffer from poor explainability as the original dimensions are lost.

1.2.1.2 FeatTS

In the FeatTS solution, we propose a clustering method for univariate time series that extract the most representative characteristics of the series. The univariate time series are temporal series where each phenomenon is measured through one single source. For this problem, most of the solutions adopt the shape of the series for discovering similarities in the data[41]. This method is powerful in very peculiar time series. Moreover, in the case of very long series, finding peculiarities requires a lot of time and memories. Our solution aims to adopt the features by converting them into graph networks to extract the inter-relationship between the signals by adopting community detection algorithms. Subsequently, a Co-Occurrence Matrix merges all the communities detected. The intuition is that if two time series are similar are often in the same community, and the Co-Occurrence Matrix permits to reveal that. We summarize our main contributions as follows:

- We introduce a novel semi-supervised clustering method leveraging the most discriminating features extracted from the time series
- Our method allows to treat at par all the features of a given dataset instead of preselecting a fixed number of them for all datasets or just leveraging the similarity of raw data.
- Our method achieves high-quality results compared with the latest baselines on the literature datasets

1.2.1.3 Time2Feat

In Time2Feat, we propose a new multivariate time series clustering. In multivariate time series, the measure of the studied phenomenon is detected through different sources. Due to

its high memory requirements, most of the solutions proposed in the literature aim to reduce the size of the series by applying algorithms for extracting features such as PCA or Neural Network. The solutions mentioned are often time-intensive and may not always provide discriminative features. Therefore, Time2Feat proposes two different feature extraction for improving the quality of the features. The first type of extraction is called Intra-Signal Features Extraction and allows obtaining features from every signal in the Multivariate Time Series. The other type of extraction is called Inter-Signal Features Extraction and permits obtaining features by considering couples of signals belonging to the same multivariate temporal series. Furthermore, both methods provide interpretable features, making possible further analysis. Through the adoption of the features, the entire process of time series clustering is more light, which reduces the time required for obtaining the final cluster. Consequently, both the solutions represent state of the art in their topic. The key contributions are summarized as follows:

- An interpretable and efficient end-to-end clustering system for multivariate time series.
- A human-in-the-loop clustering system allowing for learningbased annotations.
- A comprehensive evaluation and an open source artifacts.

1.2.2 Anomaly Detection in Streaming Time Series

Finding anomalous subsequences in time series data is a crucial problem in various contexts, from financial applications to healthcare monitoring. An anomaly might signify critical events, which is critical to reveal soonest. In recent years, many algorithms have been proposed to detect these anomalies, and most of them are based on two main techniques: Clustering and Neural Networks. The clustering is adopted for discovering if the new points obtained are placed in a known place in the space dimension, and most of the time is adopted for the offline time series. Instead, Neural Networks are usually adopted for online time series, and they forecast the future points that will be compared with the original. The original points are considered anomalous if the distance between the points is high.

1.2.2.1 Limitation of Current Approaches

Most of the approaches of Anomaly Detection adopt average or variance for computing all the previous data points and fix a tolerance threshold for the anomaly. If an observation exceeds the threshold, the algorithm marks it as an anomaly. These techniques are computationally efficient, but they do not work for most online time series as they mostly ignore the temporal aspects of the data.

The Distance Approach is another similar approach that computes the difference between the value of the last observations with the new one.ì, and if the difference exceeds the tolerance, the algorithm notifies the presence of an anomaly. However, this method can perform well only in stationary time series data because the difference between the last observation and the new one is expected to follow a consistent pattern.

Clustering algorithms project the observation in a multidimensional space and then compute the density of the observation in the space that helps to detect point anomalies. However, this method does not reveal contextual anomalies, i.e., when the point is different from the value expected, but its spaced area is highly dense.

The approach most used in anomaly detection algorithms is certainly the prediction. The time series teaches a neural network model to forecast the distribution of the following points. The main problem of this approach is the number of observations that the model needs to understand the distribution. Indeed, most Neural Network requires a high demand of observations to recognize perfectly the following points. Therefore, adopting this approach to streaming data requires much time to be accurate.

1.2.2.2 AnomalyFeat

In AnomalyFeat, we propose an algorithm for revealing anomalies from the univariate time series. The characteristic of this algorithm is the ability to work among online time series, i.e., each value of the series is obtained in streaming. Pursuing the previous solutions, we adopt the features for revealing anomalies in the series. With AnomalyFeat, we unify the two most popular algorithms for anomaly detection: Clustering and the Recurrent Neural Network. We aim to discover the density area of the new point obtained with the clustering. Instead, we adopt the Recurrent Neural Network to predict the new point, whether it will be similar to the real one, and whether both have been placed in the same cluster.

1.3 Outline

The rest of the thesis is organized as follows.

- **Chapter 1** introduces basic definitions and taxonomy related to Time Series mining. Then, we present the most adopted methods for solving the leading data mining tasks on temporal data.
- **Chapter 2** reviews the state-of-the-art of each task presented in the previous chapter.

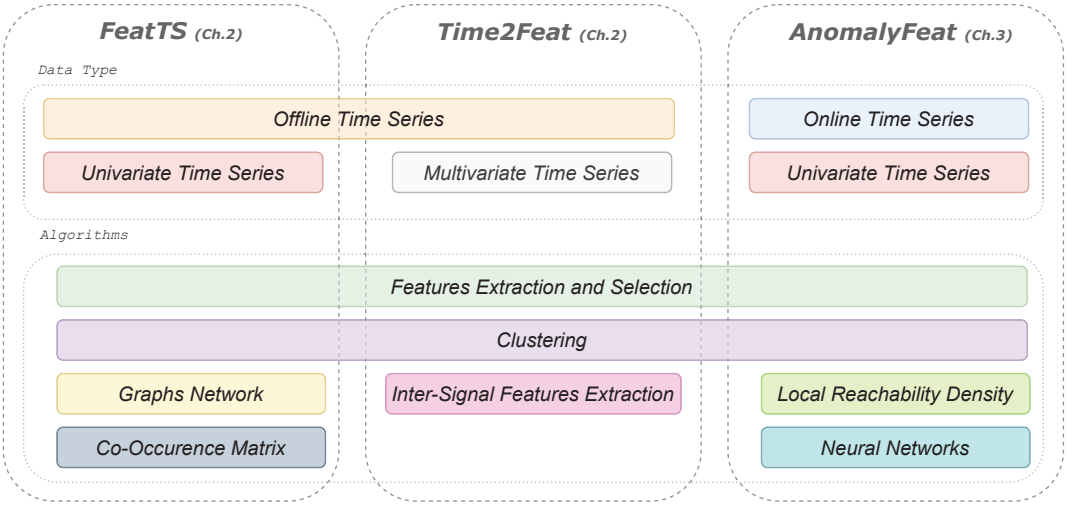


Fig. 1.2 Resuming of the Data Types and Algorithms adopted in this thesis.

- **Chapter 3** presents the techniques proposed for clustering offline time series by leveraging interpretable features. We first introduce FeatTS, a clustering method for univariate time series. This method leverages the interpretable features for creating a graphs network for extracting the global similarity among the time series. Then, we introduce Time2Feat, a clustering method for multivariate time series. This method aims to extract interpretable features by analyzing couples of signals.
- **Chapter 4** presents the technique proposed for revealing anomalies in online time series. First, we introduce some background of the task by defining the principal properties to respect. Then, we present AnomalyFeats an algorithm that combines the two main techniques for detecting anomalies(Clustering and Neural Networks) by leveraging the interpretable features.

1.4 List of Publications

- Donato Tiano, Angela Bonifati, and Raymond Ng. 2021. FeatTS: Feature-based Time Series Clustering. In Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21). Association for Computing Machinery, New York, NY, USA, 2784–2788. https://doi.org/10.1145/3448016.3452757
- Donato Tiano, Angela Bonifati, Raymond Ng. Feature-Driven Time Series Clustering. 24th International Conference on Extending Database Technology, EDBT 2021, May 2021, Nicosia, Cyprus.

- Tiano Donato, Bonifati Angela, Ng Raymond. (2021). Human-Centered Clustering for Time Series Data. Data Science with Human in the Loop @ KDD'21
- Angela Bonifati, Francesco Del Buono, Francesco Guerra, and Donato Tiano. Time2Feat: Learning Interpretable Representations for Multivariate Time Series Clustering. VLDB 2023 (To Appear)

Chapter 2

Background

2.1 Introduction

A time series is a result of observing an underlying phenomenon. Values are collected from measurements made at uniformly spaced time instants. The values are usually stored with timestamps to detect the exact moment when they are measured[145, 46]. This information is crucial because the sampling frequency can change over time, and with the timestamps, we can see when the value was obtained. The timestamp is unnecessary when the sampling frequency is constant, and the first sample is saved. The observations to study a phenomenon can come from different types of sources. Often, the number of sources defines the macro category of the time series. If the source is only one, these time series are called Univariate Time Series(UTS). Instead, if the source is more than one, the time series are called Multivariate Time Series(MTS).

Definition 1 (Univariate Time Series)**.** *A univariate time series u is an ordered sequence of values* $u = (t_1, t_2, \ldots, t_N)$*, where N is the length of the series, and* $t_i \in \mathbb{R}, \forall i$*.*

Definition 2 (Multivariate Time Series)**.** *A multivariate time series M is a set of univariate time series (a.k.a.* signals*). In particular,* $M = (u_1, u_2, \ldots, u_S)$*, where S is the number of signals, and* $u_j = (t_{j1}, t_{j2}, \ldots, t_{jN})$*. More generally, a multivariate time series can be represented as a matrix* $\mathbb{R}^{NxS}$*, where the signals are described as column vectors.*

Definition 3 (Time Series Dataset)**.** *A dataset D of time series is a set of V time series* $D = (TS_1, TS_2, \ldots, TS_V)$*. This is a generic definition,* TS_i *can be represented by an Univiariate or Multivariate Time Series.*

The peculiarity of the time series data is its capacity to exhibit a variety of patterns, each representing an underlying category[102]. These categories, also called components, help

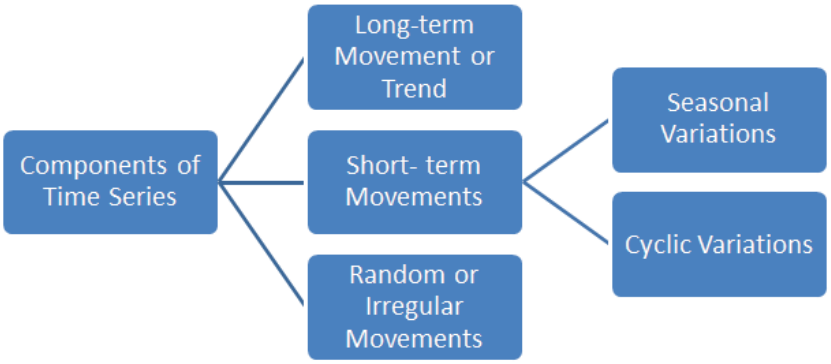


Fig. 2.1 Components Category

split the time series to obtain more knowledge about the data. Time series can include some or all of the following components: Trend(**T**), Cyclical(**C**), Seasonal(**S**) and Irregular(**I**). These components may be combined in different ways. Most of the time, they are multiplied or added:

$$u = T \times C \times S \times I \quad \text{or} \quad u = T + C + S + I$$

The components are usually divided by the length of the pattern, as shown in Figure 2.1.

Trend(**T**)[88, 135] is a long-term pattern of the time series. It represents a relatively smooth, steady, and gradual movement of a time series in the same direction. Indeed, when a time series shows a general upward pattern, it is called an uptrend, and when the trend exhibits a lower pattern, that is a downward trend.

Seasonal(**S**)[126] fluctuations describe any regular variation with a period of less than year. This variation will be present in a time series if the data are recorded hourly, daily, weekly, or monthly. These variations come into play either because of natural forces, such as seasons and climatic conditions, or artificial conventions, such as festivals and habits.

Cyclical(**C**)[89] variation is a non-seasonal component that varies in a recognizable cycle. In this case, the duration of a cycle depends on the type of event being analyzed. This cyclic movement is sometimes called the 'Business Cycle'.

Irregular(**I**)[71] variation is the unpredictable component. These fluctuations are unforeseen, uncontrollable, unpredictable, and erratic.

2.2 Approaches for time series analysis

This section summarizes the tasks that have attracted the attention of researchers in time series mining. It is organized into three main categories: Shape-Based, Features-Based Approaches and Model-Based Approaches. The shape-based approach is probably the most known and used method. This category of algorithms uses the raw data to analyze the time series and perform their main tasks analysis. Conversely, feature-based approaches adopt meaningful and interpretable characteristics for performing the principal tasks. A third category for analyzing the time series is called the model-based approach. These approaches aim to generate models or a mixture of underlying probability distributions and time series are considered identical when the models characterizing individual series are similar. Model-based approaches are not relevant for the solutions proposed in this thesis. For such a reason, they will not be discussed further in this chapter.

2.2.1 Shape-Based

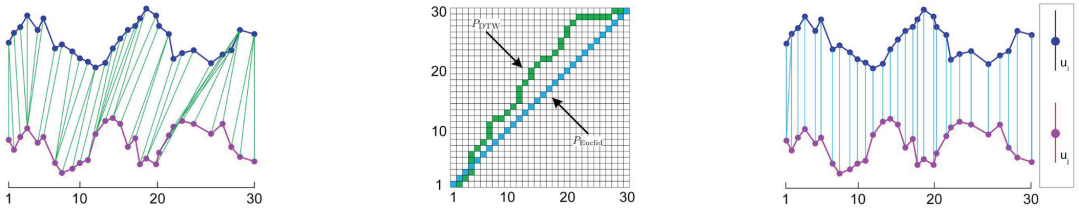
In most of shape-based tasks, computing the similarity between the time series is an typical operation[6, 37].

Definition 4 (Time Series similarity measure)**.** *A similarity measure* $\Delta(u_i, u_j)$ *of time series* u_i, u_j *is a function taking two time series as inputs and returning the distance between these series.*

The methods for computing the similarity are numerous[137, 138, 83]. The most common and easy similarity measure is the Euclidean Distances(ED).

$$ED_{u_i,u_j} = \sqrt{\sum_{k=1}^{k} (u_{i_k} - u_{j_k})^2} \tag{2.1}$$

It considers the time series a data vector and provides an exact mapping between each time point of the two time series. The complexity of this similarity measure is $\mathcal{O}(n)$, where n is the length of time series. The main problem with this measure is the lack of consideration of temporal, which can generate several pitfalls[68, 36]. Dynamic Time Warping (DTW)[14, 90] is an elastic similarity measure for sequential data. The DTW warps the time axis by finding an optimal alignment between two time series. Specifically, DTW aims to find the shortest warping path as the optimal alignment. A warping path P is a set of contiguous matrix indices defining a mapping between the time points in two time series. The number of warping paths obtained by the measure can be enormous. DTW selects the path that minimizes the global



(a) DTW Alignment (b) Warping Paths (c) ED Alignment

Fig. 2.2 Visual Representation of the DTW and ED Warping Paths.

warping cost/distance. Fig. 2.2 shows the visual illustration of the differences between DTW and ED for better understanding. Compared with ED, DTW better extracts the points with similar geometric shapes, improving the accuracy of the distance (similarity) measure. The complexity of computing DTW is $\mathcal{O}(m*n)$, where m and n represent the length of each sequence. Many techniques have improved efficiency performance in the last years. These strategies aim to restrict the number of allowed warpings. The highest acceptable distance between any couple of indexes is defined to lower bound the warping path. For instance, the FastDTW[118] calculates the warp path in a multi-resolution manner, which recursively projects a warped path to a higher resolution and then refines it. This solution allows a linear-time computation of DTW but with an exchange of information loss, leading to an optimal warping path.

2.2.2 Feature Based

The usage of the raw series for analyzing the time series is the most adopted method. This solution has proven a highly discriminating power and a good performance on most of the algorithms exposed. However, this approach profoundly depends on the quality of the raw series. Indeed, most algorithms compute the distances between the series or series subsequences to discriminate the similarity. Therefore, this approach is fruitful when the series presents discriminant shapes or apparent differences between each series. Moreover, another problem with the raw series is the quality of the sensor that has recorded them. Many low-quality sensors could miss some data points during the recording of the series or, even worse, can produce noisy data. These inaccurate values can prejudicate the performance of the algorithms. Another problem that arises when the raw series are adopted is time performance. Computing the distance between long time series is time consuming. Moreover, in the case of a large dataset, producing the distances between each series can harm time performance. Moreover, the longer the series are, the more the distances increase, making it challenging to distinguish a series from another and reducing the quality performance. Besides performance

problems, having very long time series causes an increase in the required memory space. Adopting the distance for measuring the difference between the time series in some domains is not discriminatory enough. This characteristic alone is probably not enough to obtain a good analysis of the series. In the last years, it has been proven that a high number of uncorrelated characteristics improve the quality of the task analysis. Moreover, with the explosion of the Explainable AI (XAI)[119] trend, adopting explainable characteristics can produce deep analysis among the subject domain.

A possible solution that has been proposed to solve this problem is to extract high-level view of the time series rather than only their raw shapes and distances. The idea is to capture different properties from same series and analyze the temporal sequence by studying these properties. This solution is known under the name of Features Analysis of time series. The features, or characteristics, allow to capture a global picture of the data, identifying different details of the series and with different natures. The process for detecting and extrapolating the characteristics of the time series is called Features Extraction. This procedure allows to summarize the entire series in a vector of values, where each value represents the extracted feature. The first problem that the adoption of the features permits to solve is connected with its property of summarizing long and noisy time series. Indeed, the features are less susceptible to the noisy time series because of their statistical nature, allowing them to balance the noise with the entire series. Moreover, the feature extraction can be considered a dimensionality reduction operation. In some cases, the summarized characteristics of the time series can provide a more meaningful dimensionality reduction than other methods. Indeed, many features that reduce the dimensionality of the series are interpretable, i.e., adopt some interpretable formulation for summarizing the entire series. This property of the features is significant for knowing the nature of the features that better interpret the domain of the time series. There is extensive literature on time series analysis methods that characterize time series properties. The easiest way to categorize the features is by their nature. The distribution features are probably the most straightforward measure of the time series values. They are independent of the time ordering of the points but can express inherent information of the series. Examples of distribution features are mean, variance, distribution entropy, and measure for outliers. An example is the absolute energy of the time series, which is the sum of the squared values.

$$E = \sum_{i=1}^{n} x_i^2 \tag{2.2}$$

In the formula, x_i represents the values of the time series and n is the length of the series.

The autocorrelation features allow the measurement of the correlation between time series values separated by a given time lag. These features are very useful for finding repeated signals in the presence of noise. The autocorrelation is computed as follows:

$$C(\tau) = \frac{1}{s_x^2(N-\tau)} \sum_{t=1}^{N-\tau} (x_t - \bar{x})(x_{t+\tau} - \bar{x}) \tag{2.3}$$

In the formula, τ represents the time lag for the time series x, with variance s_x^2 and mean $\bar{x}$.

The family of Entropy Features are used to quantify the grade of predictability in a time series. These features are based on Information Theory, a branch of mathematics that permits to discover and explore the mathematical laws underlying the behavior of data. Some examples of the Entropy Features are Approximate Entropy (ApEn)[104], Sample Entropy (SampEn) [112], and Permutation Entropy (PermEn)[50]. The PermEn quantifies the complexity of a dynamic system by capturing the order relations between values of a time series. Moreover, it extracts the probability distribution of the ordinal patterns, i.e., the frequency of occurrence of each possible ordinal pattern in a time series of discrete values. The PermEn does not require any parameter in input, and it is very robust to the noise and the time complexity. The formula of PermEn is expressed as:

$$PE_{D,norm} = -\frac{1}{log_2 D!} \sum_{i=0}^{D!} p_i log_2 p_i, \tag{2.4}$$

In the formula, D represents the embedding dimension, i.e., the size of the subsequence to permute. The parameter *pi* represents the relative frequency of each permutation. This value is computed by dividing the number of times the permutation is found in the series by the total number of sequences. The formula is normalized between 0 and 1. As the values decrease, the time series tends to exhibit greater regularity and determinism. Conversely, when the values approach 1, the time series is more likely to be noisy.

Another family of features is based on the extraction of a set of models among the series. The idea is to find the most suitable parameters for each model proposed and use these parameters to compare the series. Most of these models are based on the regression model, such as ARMA, ARIMA, GARCH, or statistical models, such as the Hidden Markov Model (HMM) and Gaussian process models.

The Regression and Statistical models are not the only ones used for extracting the features among the series. The non-linear time series analysis methods, including embedding dimensions and fluctuation analysis, allow to obtain non-linearity measures, such as the Fractal Dimension Spectrum of a time series[77].

The Stationary Features allow to quantify how properties of a time series change over time. For example, the Average Stationarity metric (StatAv) provides a measure of mean stationarity

$$StatAv(\tau) = \frac{\mu(\{\overline{x_{1:w}}, \overline{x_{w+1:2w}}, ..., \overline{x_{(m-1)w:mw}}\})}{\mu(x)} \tag{2.5}$$

This metric computes m non-overlapping windows, with a length of w, among the time series. A standard deviation μ is computed among each window, and time series in which the mean in windows varies more than the entire time series have higher values of StatAv.

The Fourier transform in discrete time allows a time series to be represented as a linear combination of frequency components, such as power spectrum, wavelet spectrum, and other periodicity measures. The most representative formula is expressed as:

$$\tilde{x_k} = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} x_i e^{\frac{2\pi nki}{N}} \tag{2.6}$$

In the formula, the real and complex parts of $\tilde{x_k}$ encode the amplitude and phase of that component, $e^{\frac{2\pi nki}{N}}$, for frequencies $f_k = \frac{k}{N\Delta t}$, where Δt is the sampling interval.

Other basis function decompositions use a wavelet basis set under variations in temporal scaling and translation to capture changes. Those linear systems of equations can only produce exponentially growing (or decaying) or (damped) oscillatory solutions. However, irregular behavior in a linear time series must be attributed to a stochastic external drive to the system.

Other properties like extreme events[4] or visibility graphs[70], and other statistics from biomedical signal processing can be applied to a time series for obtaining features from different domains. For example, many features can be derived from the heart rate variability (HRV) literature[81].

With the explosion of statistical methods for analyzing the time series, the number of features for exploring the temporal sequence is augmented. The literature presents studies[8, 30, 38] where the number of possible features extractable from the time series can surpass 7000 for each series. Therefore, the selection of which features can be relevant for capturing the approximation of the dataset represents the fundamental problem of these methods. Timmer et al. wrote: “The crucial problem is not the classificator function, but the selection of well-discriminating features. In addition, the features should contribute to an understanding [...]"[133]. The choice of the features for characterizing the time series is often based on the subjectivity of the data analyst. The problem with this solution is in the ability of the analyst to know the domain and select the most suitable features. Moreover, it is not easy to know whether the features used by one researcher would work better than

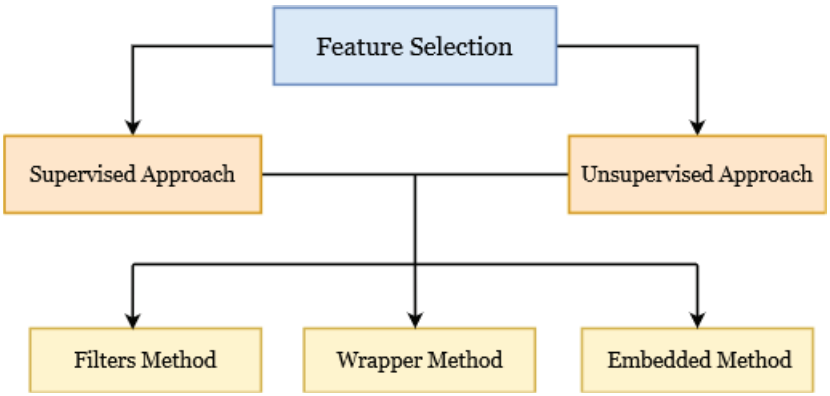


Fig. 2.3 Taxonomy of Features Selection Methods.

others. This human methodological comparison is complicated to perform because of the vast and interdisciplinary character of the time-series analysis literature and the different nature of the extractable features.

In the last years, many techniques have been developed to select the best features automatically. In the machine learning field, this operation is often divided into two categories Supervised and Unsupervised techniques. Figure 2.3 shows a taxonomy of the primary ways to select the features in both categories.

In the **Filter Method**, features are selected based on statistics measures. It removes redundant and irrelevant features by ranking them. This ranking is computed by correlating the features with the output values. In the supervised approach, the review is done by checking if the features and the labels are positively or negatively correlated.

In the **Wrapper Method**, selecting the features is considered a search problem. The dataset is split into different subsets of features and evaluated. The set of features that obtain the best performance is provided in the output.

The **Embedded Methods** combine the advantages of filter and wrapper methods by considering the interaction of features along with the low computational cost.

In the filter category, the Benjamin–Hochberg[13] procedure is a semi supervised method that evaluates the importance of every single feature by applying a multiple testing procedure to decide which features to keep and which to delete. The multiple testing is supervised, and it is based on two hypotheses:

$$H_0 = Target\ and\ Feature\ are\ independent.$$
$$H_1 = Target\ and\ Feature\ are\ dependent.$$

where indipendent means that the feature does not influence the target. Viceversa, the features influence the target.

The Principal Features Analysis (PFA)[82] is an Unsupervised Method based on the Principal Component Analysis (PCA). PCA is one of the popular methods used and can be shown to be optimal using different optimality criteria. However, it has the disadvantage that all original features are used in the projection to the lower-dimensional space. This projection is the critical difference between PFA and PCA. Indeed, PFA preserves the original values of the features and thus the distance between them. The only parameter required by the PFA is the Explained Variance(EV). This value represents the ratio between the variance of one single feature and the sum of variances of all individual features. PFA computes the EV of each feature and orders it by the highest ratio. Finally, PFA takes the minimum number of the first F features which the sum of their EV is higher than the fixed Explained Variance.

Recursive Feature Elimination(RFE)[29] is the most supervised method used in the Wrapper category. RFE aims to select features by considering smaller and smaller sets of features. The algorithm trains a Support Vector Machines (SVMs) model to obtain each feature's importance through any specific attribute. The least important features are pruned from the last set of features. The procedure is repeated on the pruned set until the desired number of features to select is eventually reached. [19] uses a new optimization criterion for minimizing the intra-cluster and maximizing inter-cluster inertias. The authors propose a function based on minimization-maximization of the variance of scattering matrices obtained from the clusters built by the k-means clustering algorithm. This function assigns a ranking score to each partition that may be defined in the search space of all possible subsets of features and the number of clusters. The criterion proposed in this method provides both a ranking of relevant features and an optimal partition.

Finally, in the Embedded Features Selection, the state of the art of Supervised approach adopted for selecting the features is the Sparse Multinomial Logistic Regression(SMLR)[66]. SMLR is an implementation of a sparse regularization of the Automatic Relevance Determination (ARD)[146] of the classical multinational logistic regression. The ARD is based on a Bayesian Ridge Regression and estimates the importance of each feature. Finally, the SMLR prunes the not-valuable features for the prediction.

A Unsupervised Embedded Features Selection was proposed in [127]. In this method, the authors combine spectral feature selection and the Calinski-Harabasz index for selecting a relevant feature subset. The feature selection operates in two stages: in the first stage, the algorithm identifies the features that keep the data structure by computing the Laplacian Score for each feature. In the second stage, feature subsets are evaluated through a modified

internal evaluation index called WNCH (Weighted Normalized Calinski-Harabasz index). The features selected correspond to the subset that has the highest WNCH value.

2.3 Tasks

This section presents the principal tasks in time series analysis. The tasks have many common notions, but for simplicity's sake, it is organized into four different objectives.

2.3.1 Similarity Search

The problem of similarity search is to find a subset of items that are the nearest to a query item, called nearest neighbors, under some distance measure from a dataset[6, 37].

The K-Nearest Neighborhood(k-NN)[74, 35] is the most used algorithm for creating the list L of series in similarity search. This algorithm computes the proximity by applying a similarity measure Δ between each data value.

In the time series **Shape-Based** similarity approaches, two main categories of algorithms are proposed: The Whole Matching(WM) Queries and the The Subsequence Matching(SM) Queries. The Whole Matching(WM) Queries compute the similarity between an entire query series and an entire candidate series. Therefore, the length of the series involved has to be the same. The Subsequence Matching(SM) Queries compute the similarity between an entire query series and all subsequences of a candidate series. In this case, the candidate series can have different lengths.

For instance, in the case of time series, computing the k-NN with the Euclidean Distance as a similarity measure has $\mathcal{O}(n \times V)$ as time complexity. It represents the time complexity of the Euclidean Distance repeated for each element of dataset D with the time series query u_q.

The analysis of the series is usually made through two main methods: Sequential and Indexing methods. The Sequential approach[109] directly compares each time series of the dataset with the query series. This approach leads to an enormous amount of comparison and a massive amount of time to be completed. Therefore, many optimization algorithms improve this technique. The Indexing approach[143, 152, 121] generally has two main steps: in the first step, it reduces the number of candidates by creating a pre-built index. Subsequently, the not filtered series candidates are then compared to the query series in the original space. There also exist hybrid approaches[65] that combine indexing and sequential methods. In particular, the data are transformed and reorganized in levels in the multi-step approaches.

The filtering of the candidate's series occurs at each step, and the data are sequentially read one at a time.

Searching for a list of candidates series among many data can require much time. Moreover, this problem is enhanced when long raw time series are treated. The summarization techniques [141] permit to reduce the computational time by encoding the time series in a short representation, identifying points or observations in time series data that are relatively more important than other points in the input data. The principal techniques for summarizing the Time Series are divided into two categories:

- **Non-data adaptive** representation applies equal parameters for converting every time series in the database regardless of its nature.
- **Data Adaptive** representation applies a standard representation for all database items that minimize the global reconstruction error.

Based on the high demanding time requested by the Shape-Based approach, the **Features-Based** approaches are a potent tool for fastly searching time series. Indeed, these approaches allow to reduce the dimensionality of the temporal series and to optimize the time of the similarity search. Most of the algorithms proposed extract a set of features on time series. Then, the k-NN algorithm is applied to the extracted features for detecting the k closest series of the item query.

2.3.2 Clustering

Clustering is a data mining technique that aims to place together similar data, creating natural groups called clusters. The goal is to discover the most homogeneous groups that are distinguishable from other groups. Formally, data distribution over clusters should be carried out to maximize inter-cluster variance and minimize intra-cluster variance. In both the approaches, **Shape-Based** and **Feature-Based**, the clustering algorithms are divided into two categories: Hierarchical and Partitional algorithms. Hierarchical clustering creates a hierarchy of groups, as its name suggests. Clusters are created by merging or dividing the groups from the next lower or upper level, such that an ordered sequence of groupings is obtained. These methods are called Agglomerative or Divisive. In agglomerative procedures (bottom-up), every data member starts in its cluster. Members are grouped sequentially based on the similarity measure until all members are contained in a single cluster. Divisive procedures do the opposite, starting with all data in one cluster and dividing them until each member is in a singleton. Hierarchical clustering is one of the most powerful visualization tools for time-series clustering. For example, [101] uses agglomerative clustering to predict

the road accident data in combination with trend analysis. In another study, [48] the authors use the agglomerative clustering to study the behavior of the people's activities in the field of public transit demand analysis. Moreover, in many types of research, Hierarchical is used to evaluate dimensionality reduction or distance metric due to its power in visualization. For example, in a study [79], the authors presented Symbolic Aggregate Approximation (SAX) representation, using hierarchical clustering to evaluate their work. The results obtained show the same performance between SAX and the Euclidean distance. Another notable feature of this algorithm is that it does not require entering the cluster number as an initial parameter. Indeed, defining the number of clusters in real-world problems is very complicated. The principal weakness of the Hierarchical Algorithm derives from the poor flexibility in the creation of the clusters in the divisive and agglomerative method. Therefore, the Hierarchical algorithms are usually adopted with other algorithms[54, 86] as a hybrid clustering approach to fix this problem. In the partitional clustering, the algorithm assigns each data point of the dataset in k different clusters specified beforehand. The procedure follows combinatorial optimization problems that minimize the intracluster distance while maximizing the intercluster distance. In the most classical procedure, k data points, called prototypes, will be chosen randomly and assigned to individual clusters. Then, distances between the data points and prototypes are calculated, and each object is assigned to the cluster of its closest centroid. Subsequently, a function updates the prototypes of each cluster, and distances and prototypes are updated iteratively until a certain number of iterations have elapsed or no object changes clusters anymore. One of the most used partitioning clustering algorithms is k-Means[56], where each cluster has a prototype based on the mean value of its objects. The main idea behind k-Means clustering is minimizing the total distance between all objects in a cluster from their prototypes. In most time series clustering algorithms, the k-Means are challenging to apply. Indeed, finding a prototype that represents the time series average is a complex operation. Therefore, the most used approach for this domain is the k-Medoid(**PAM**)[100]. This algorithm is similar to k-Means, but the prototype is represented by an original data of the dataset, called centroid. The classical approach[47] for the time series clustering with the k-Medoid is to randomly choose the k time series and then adopt a similarity measures to obtain the distances between the centroid and the other time series. By definition, most clustering algorithms are entirely unsupervised (i.e., any information about the data points is furnished in advance). In the last years, many Semi-Supervised solutions have been proposed to improve the performance of the algorithms [108]. Semi-supervised clustering is a new method that combines semi-supervised learning (SSL)[151] and cluster analysis, using a small amount of prior information to process unlabeled data. The preliminary information provided to the semi-supervised clustering algorithms is usually

divided into *Independent Class Labels* and *Pairwise Constraints*. In *Independent Class Labels*, a small amount of data is furnished in input with the dataset. SeededKMeans[9] is an example of this category. The latter uses the labels provided as input to find the constraints among the data and the centroids and then applies the k-Means algorithm to find the clusters. The *Pairwise Constraints* method relies on a small number of labels of the original dataset to create two kinds of links, i.e., Must Link and Cannot Link. Must links are connections between two data points that represent a 'constraint of belonging'. This means that the data points (or time series at large) should be clustered together. Cannot Links do the opposite, thus leading to separate data points.

2.3.3 Classification

The classification task aims to assign labels to each series of a dataset. The main difference with the clustering task is the information provided in advance. Indeed, in classification, the labels of each time series are provided in advance. The purpose is to adopt the labels for learning the most prominent characteristics for distinguishing the classes. Then, when an unlabeled dataset is entered into the system, it can automatically determine which class each series. Learning and discriminating functions are usually called 'Training' and 'Testing'.

With the growth of diverse research areas, the number of approaches for classifying the time series is enormous. In literature, the approaches are usually divided by the discriminatory features that the technique attempts to find. The main approaches can be divided into Distance-Based, Interval-based, Dictionary-based, and Shapelets-based.

The distance-based is probably the most common approach. In Time Series Classification, the method for obtaining the class of the unlabeled data is very similar to the Similarity Approach explained in Section 2.3.1. The most common technique is to train a k-NN model. Each time series is categorized by its label and the algorithm assigns the class of an unlabeled series by computing the closest k series, as shown in Figure 2.4. The computation of the distances is made by computing the differences between the **Shape** or the **Features** of the series.

In the interval-based approach, subseries of the original series are extracted for classifying the entire time series. This solution is mainly adopted when the long time series is characterized by a high quantity of regions of noise that could confound the classifier. The problem of this solution is to find the correct interval. The approach adopted from many time series classification algorithms is to generate many different random intervals and classifiers on each one, ensembling the resulting predictions. This approach is mainly adopted by the **Shape-based** approaches.

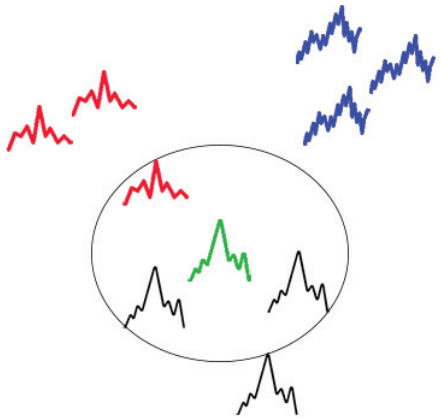


Fig. 2.4 Classification with kNN, with k=3

The dictionary-based classifiers are a family of time series classification algorithms that capture the frequency of pattern occurrences in a time series. This approach is potent when the discriminatory repeated patterns occur more frequently in one class than in other classes. Generally, a base classifier takes a window of a real-valued series, trims it to create a shorter real-valued series then discretizes it to convert it into a symbol, also called a word. Histograms of symbols are formed from all windows. Most of these approaches are adopted by **Features-based** approaches.

The shapelets-based classifiers are a set of time series classification algorithms that extract subsequences or small sub-shapes of time series representative of a class. This family of classifiers is potent in extracting unusual patterns that occasionally occur at any point during the measurement, allowing the classification of the unusual patterns within the same class. Most of these approaches are adopted by **Shape-based** approaches.

All the above approaches need heavy crafting on data preprocessing to perform well. Indeed, as seen in the shapelet-based approaches, choosing the correct shapelets is an operation that requires a high amount of time and precision. Deep Neural Networks(DNN)[61] have been employed in time series classification tasks in the last years. The primary purpose of the DNNs is to obtain complex dependency structures between the data, where this operation is highly complex in the previous approaches. The main problem with applying the time series in the DNN is their structure. Indeed, the networks have to be designed to accept the hierarchical representation of the series. A DNN is a structure composed of several parametric functions called simplicity layers. These layers are composed of 'Neurons', a structure that receives several inputs and weights and produces an output based on the activation function. The activation functions[110] are the core of the DNN[134]. They permit the neurons, and thus the layers, to communicate between them. Therefore, each layer receives the input and weights from the previous layers and produces the input for the consequent

layers. The weights have another essential rule in the DNN. Indeed, they can be adapted to adjust the input of each neuron for obtaining the desired output in the next layer. This operation is called feed-forward propagation[11]. The values of each weight are fixed during the training phase. The training phase teaches the neural network to recognize the class of each series. In input, a high amount of series is provided with their class of belonging. The weights are randomly assigned above all the layers in the first iteration. At the end of the first iteration, the network produces a vector in which, for each series belonging to the dataset, it is estimated the probability of belonging to each class provided in the input. These estimations are compared with the original classes of the series, where a cost function computes the loss between the original class and the estimated class. Then, the backward pass propagates[49] the error on each neural network weight. The new values of the weights are estimated by adopting the gradient descent technique that permits finding a local minimum/maximum of a given function, which in this case is to minimize the cost function. Therefore, repeating the feed-forward and the backpropagation permits minimizes the estimation error and improves the neural network's performance. This structure is a simple representation of a neural network, and it is usually indicated as Multilayer Perceptron(MLP). Once the loss function is minimized, the neural network can classify the new series. This process is called Testing, and it is similar to the process of feed-forward. Indeed, a series is given in input to the neural network that produces a probability estimation of each class. The class with the highest probability is assigned to the series.

Although many types of DNNs exist, the most adopted in time series classification is the Convolutional Neural Network (CNN)[3]. This DNN has been successfully used in many domains to recognize objects in different contexts, but its power has also been shown in class recognition. The CNN is composed of the repetition of two operations: Convolution and Pooling. The Convolution operation flows some convolution filters on the raw time series. These filters are usually determined previously according to domain knowledge or based on experiments. Therefore, this operation permits filtering redundant information. The Pooling operation is another filtering operation that permits down-sampling of the convolutional output, reducing variability in the hidden activations. This operation is made by reducing the results of each convolutional filter by computing the average of each filter or taking the maximum value. After several convolution and pooling operations, the original time series is represented by a series of feature maps. Then the features are connected to a multilayer perceptron (MLP) to perform classification. Figure 2.5 shows an example of time series classification adopting the CNN when the shape is used[72].

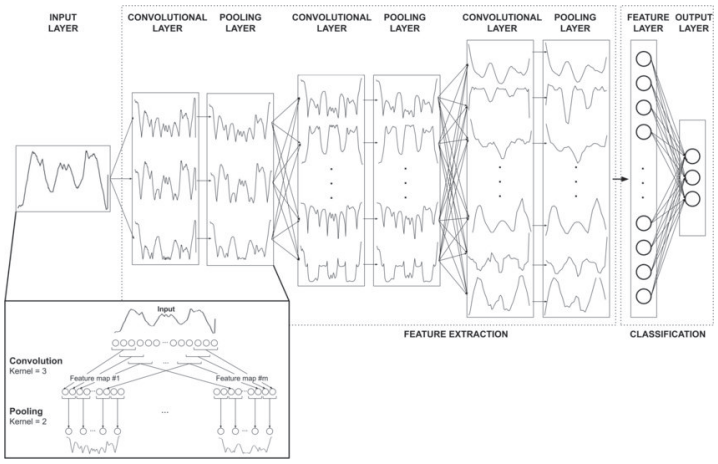


Fig. 2.5 Time Series Classification with CNN on Raw Time Series

2.3.4 Anomaly Detection

The anomaly detection task aims to find unexpected events in data. By definition: “An anomaly is an observation or a sequence of observations that deviate remarkably from the general data distribution. The set of the anomalies forms a tiny part of the dataset"[18]. This task has been a research topic for a long time, but it has gained more attraction in the last years. The motivation derives from the augmented digitalization data that makes the automatization of the detection anomalies necessary. Indeed, various domains, such as health care, fraud detection, and intrusion detection, have outsourced the search for possible anomalies to automated anomaly detection techniques. A necessary distinction has to be made between detecting the anomalies and the noise. Indeed, these two operations strongly depend on the interest of the analyst or the particular scenario considered. The noise detection is orientated on cleaning/correcting the data, and any analysis is made once the noise is detected. The principal objective of the noise revelation is to improve the original dataset and increase the algorithm’s performance that adopts the dataset[117]. By contrast, in anomaly detection, the objective is to comprehend the anomalous event and the cause. An important aspect enhanced from the definition is the number of anomalies expected in the dataset. An anomaly is defined as an infrequent and unexpected event, meaning that the number of normal data points is significantly higher than the number of anomalous points.è This information is crucial in creating a methodology for discovering anomalies because some standard techniques require a balanced dataset to work well. In time series data, an anomaly can assume different forms. These forms can be resumed in three categories:

- **Point Anomaly**: When a datum deviates significantly from the rest of the data (global anomaly). This anomaly represents the easiest irregularity to reveal. Figures 2.6 shows

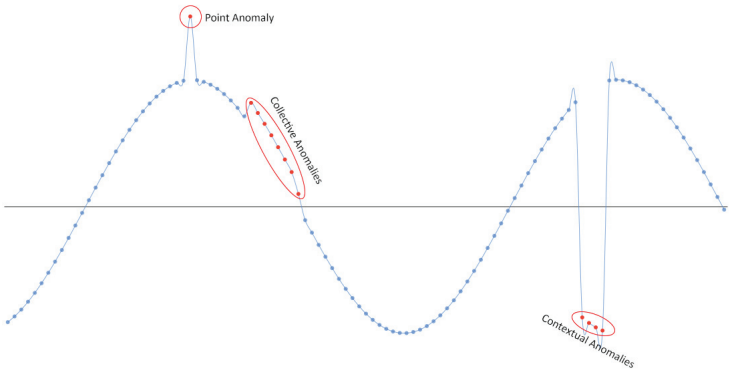


Fig. 2.6 Type of Anomalies

an example of this anomaly. This anomaly could represent a credit card transaction that is tremendously higher compared with the usual.

- **Collective Anomalies**: A collection of continuous points shows a different pattern from standard data over time. In this form, the points that belong to the suspicious pattern may not be anomalies by themselves, but their occurrence together can be identified as anomalous. Figures 2.6 shows an example of this anomaly. This anomaly could represent a credit card transaction that could be usual if done in a single day but suspicious if it is repeated for more than two days in a row.

- **Contextual anomalies**: When a point or collective instance is suspicious in the context when they arise but normal in some other context, it is defined as Contextual Anomaly. This type of anomaly arises when the dataset has a clear structure, i.e., it is specified in the problem formulation. Figures 2.6 shows an example of Contextual Anomalies.

The approaches proposed for solving this problem are often divided into statistical, classical machine learning, and neural networks approach. Most of these approaches are Shape-based. Indeed, detecting anomalies by adopting a Feature-Based approach is still an unexplored domain. In the statistical approach, the most well-researched techniques are based on the regressive models, such as AR, MA, ARMA, and ARIMA. These models detect the anomaly by computing the expected point or subsequence point and comparing them with the original. The more the data are different, the higher the probability of detecting an anomaly.

The Autoregressive Model **(AR)** is the basic regression model for detecting an anomaly. This stochastical approach permits forecasting a dependent variable value X_t on a finite set of

previous independent variable values of length p and with an error value ε:

$$X_t = \sum_{i=1}^{p} a_i \cdot X_{t-1} + c + \varepsilon_t \tag{2.7}$$

The coefficients a_i, and c are computed by training the data and solving the corresponding linear equations with least-squared regression. The error coefficient ε_t represents the anomaly score, computed by the difference between the forecasted and observed values. This model works appropriately only on stationary data, making necessary a transformation if the dataset does not respect this property.

The Moving Average**(MA)** computes a linear combination differently from the AR. MA computes a linear combination on the last q predicted error $\{ \varepsilon_t, \varepsilon_{t-1}, ..., \varepsilon_{t-q} \}$, rather than the last p observation as AR.

$$X_t = \sum_{i=1}^{q} a_i \cdot \varepsilon_{t-1} + \mu + \varepsilon_t \tag{2.8}$$

In the definition, μ is the mean of the entire time series, and the coefficients $\{a_0, ..., a_q\}$ are learned from the data. The main problem with the MA is to obtains the q predicted errors. These values are usually obtained once the model is fitted. Therefore, the only way to find the X_t is to optimize the error sequentially, leading to the time performance of the algorithms. Once the model is fitted, the procedure for detecting the anomaly is equal to the AR model.

The power of these two models is the capability to be combined with other models. ARMA is, for example, the combination of AR and MA. The linear model depends on the p last observations and the q last errors with this model.

$$X_t = \sum_{i=1}^{p} a_i \cdot X_{t-1} + \sum_{i=1}^{q} b_i \cdot \varepsilon_{t-1} + \varepsilon_t \tag{2.9}$$

The sticking point of this model is the choice of the p and q coefficient. Adopting a higher number of p and q, the model uses many points for converging. The result is an overfitted model that can produce many false negatives during the anomaly evaluation. Conversely, if the model is trained on low values of p and q, it can produce an underfitted model, producing many false positives. Many algorithms have been proposed for fitting the models and finding the correct values of p and q [1, 17, 59].

Combining the two models, AR and MA do not solve the AR model's problem of treating the non-stationary time series. A model called ARIMA solves this problem by adopting the differencing, a method that permits making a non-stationary time series stationary. The idea

Fig. 2.7 Example of Differencing Operation

is to compute the difference between consecutive points to remove some trends from the data. Formally, the differencing is defined by a parameter d which defines the number of times the time series is differenced. For example, with $d = 1$, each point of the time series is differenced from only the previous point.

$$X_i = X_i - X_i', \forall i \in \{1, .., T\} \tag{2.10}$$

Figure 2.7 shows an example of non-stationary data. Setting the differencing parameter d equal to 1 and 2, the time series produced is shown in Figure 2.7. This model detects the anomaly as the previous models exposed.

Other methods for discovering anomalies in time series derives directly from the task previously exposed. The clustering is probably the most adopted solution for discovering anomalies in time series. KMeans, a partitional clustering exposed in Section 2.3.2, suits well for exploring the time series and discovering anomalies among the data points.

The growth of systems capable of recording the data in real-time has augmented the necessity of detecting anomalies in streaming time series. This new data type has increased the difficulty of revealing the anomalous point. Indeed, the techniques that aspire to detect anomalies in streaming data have to respect other constraints:

- **Transient:** In the data stream, the importance of a point is directly proportional to the time elapsed since it was detected. It means that an outlier detection algorithm should detect outliers of the observation immediately as it arrives.
- **Infinite:** There is an endless reception of points in data streaming by nature. A necessary consequence is an impossibility of keeping the entire dataset received in memory. Therefore, all the solutions requiring the whole dataset to detect outliers cannot be adopted. It means that the dataset needs to be summarized, and when a new point is received, it has to be compared with the summarization of the previous points.
- **Arrival Rate:** This constraint is strictly correlated with the Transient constraint. This constraint binds the outlier detection technique for the data stream to process data points before the next data point arrives. This limitation is not easy to ensure because the reception of each point in the data stream can be fixed or variable. Therefore, the detection of the anomalies should be adjusted based on the available processing time.
- **Concept Drift:** The endless reception of the points of the data streaming can affect the distribution of observations by drifting the trend. Therefore, the anomaly detection algorithm must automatically adapt to this drift, assuming any fixed distribution of data.

There are many approaches in the literature for solving anomaly detection algorithms on Online Time Series data. The most straightforward approach adopts some statistical operators as average and variance. The idea is to compute the mean and variance for all the previous data points and fix a tolerance threshold to the anomaly. If an observation exceeds the threshold, the algorithm marks it as an anomaly. This technique is very computationally efficient in terms of time and memory requirements; however, these approaches do not work for most time series as they mostly ignore the temporal aspects of the data. Moreover, they cannot detect a majority of contextual and collective anomalies[84]. Another similar approach, called Distance Approach, computes the difference between the value of the last observations with the new one. Like the previous approach, if the difference exceeds the tolerance, the anomaly detection algorithm notifies the presence of an anomaly. This method also seems to be relatively straightforward. Indeed, it can perform well only in stationary time series data[26].

The clustering approach exposed for the offline time series anomaly detection is also applicable to the online anomaly detection. The idea is essentially based on the computation of the Density Area around a point. However, classical clustering is complex to apply to the time series streaming because most of the solutions must keep in memory all the

points obtained, but this does not respect the Infinite constraint. Moreover, clustering is not applicable in the presence of contextual anomalies. Indeed, if an instance is locally anomalous, i.e., different from the value expected, but globally the instance is known, the clustering does not detect it as an anomaly [106, 64].

The approach most used by the real-time anomaly detection algorithms is undoubtedly the Predictive. With this approach, the time series teaches a regression model[39] to forecast the distribution of the next point. In this way, the algorithm compares the subsequent observations with the forecasted distribution and, in case the deviation is high, it will mark them as anomalies.

This approach is probably the most complete compared with the previous. Indeed, it permits handling all the kinds of anomalies discussed before. The main problems of this approach depend on the number of observations that the model needs to learn the distribution. Teaching regressive statistical models, like ARIMA[139], or even worse, the Regressive Neural Network, like LSTM[43], to recognize perfectly the following points requires a high demand of observations. Therefore, adopting this approach among streaming data requires much time to be accurate, violating the Transient constraint.

Finally, the Autoencoder approach now appears the trending topic. With the Autoencoder, the input data passes into an Encoder algorithm, which tries to summarize the input data as much as possible, creating an object called Code. This Code is the input of another algorithm of the Autoencoder, called Decoder. The Decoder aims to recreate the initial data input based on the object Code, minimizing the difference between original and recreated input[40]. Encoder and Decoder are frequently Neural networks. Indeed, the Encoder is a multilevel neural network, where the number of neurons per level decreases as the levels increase to reduce the number of features required. The final layer outcome represents the Code. The Decoder is even a neural network with the difference that the number of neurons per level increases as the levels increase to reconstruct the initial input [28].

This operation permits recognizing an anomaly when the output of the Decoder is mainly different from the Encoder's input[69]. The reason is if there is an anomaly inside the input data. The Encoder creates a different Code from the ones created during the training. Thus, the Decoder tries to decode a never-seen Code. Therefore, the expected output of the Decoder will contain many differences from the original input. Hence, this mistake in reproducing the initial input depends on an anomaly of the data in the input.

The results of this approach are exciting, but most of them require much data to train perfectly the Encoder and the Decoder. Moreover, this method appears like a black box, making a deep analysis of the anomalies impossible.

Another strategy adopted in the last years is to use an Ensemble approach. The idea is to use a variety of algorithms to control each observation and apply some form of voting mechanism [107] for the output of each algorithm. An ensemble setup can be composed of similar algorithms, such as a set of Clustering approaches, or with different combinations of anomaly detectors, such as Statistical and Predictive Approaches.

This method can improve the performance of the other solutions, but it increases the configuration complexity and the computational time. This increase derives from the manner the algorithms are combined. Indeed, in literature, most ensemble approaches execute the outlier detectors in parallel, without any influence between them. Subsequently, the voting system has to study each evaluation and then decide. Moreover, this operation requires heavy computation approaches like Predictive and Clustering, a very high time complexity for good performance.

Chapter 3

Related Work

3.1 Similarity Search

In the **Shape-Based** approach, Piecewise Aggregate Approximation (PAA)[67] is a non-adaptive technique that approximates a time series by dividing it into equal-length segments and recording the mean value of the data points within the segment. In addition, [87] proposes Multi-Resolution PAA, which constructs the representation at various resolution levels. Instead of using mean values, it adopts the segmented sum of variation (SSV)[73] to represent each segment. At the same time, proposes extracting and concatenating each local segment's amplitude-levels local features (ALF) to represent the whole series. In general, most of the non-adaptive techniques are generally less efficient than the data adaptive approaches as they do not adapt to each data[141]. Adaptive Piecewise Constant Approximation(APCA)[25] is an adaptive summarization technique that transforms each time series into a set of constant value segments of various lengths based on its minimal reconstruction errors. Another method, called Piecewise Linear Approximation (PLA)[123], is a widely used representation for the time series summarization task. Each linear segment in PLA is represented by polynomial coefficients and can be obtained either by interpolation or regression. Another method for summarizing the time series is to transform the sequence into a set of symbols. Symbolic representations authorize to take advantage of the richness of data structures and algorithms in the text processing and bioinformatics communities. The Symbolic Aggregate approXimation(SAX)[79] is probably the most known algorithm of symbolic representation. SAX first transforms the data into the PAA representation and then symbolizes the PAA representation into a discrete string. In this way, the symbolic distance measure defined in SAX would allow lower bounding of the PAA distance.

By contrast, an algorithm that leverages the **Features** for querying the time series in a dataset is TimeExplorer[32]. TimeExplorer(TE) extracts a small number of characteristics

among the raw time series. These features include the classic distribution characteristics, such as means and standard deviations. Moreover, some other features have been discovered individually by other researchers, such as sharp increases[125], mountains[22], and serial periodicities[24]. The TE implements three analysis tasks:

- Brushing: The user provides a time series, and TE retrieves all similar time series in the database.
- Sorting: Sort time series based on their relevance to the time series provided.
- Filtering: Extracts all the time series that satisfy filtering conditions on their feature space provided by the user.

These operations are based on a Weighted Dissimilarity computation. Indeed, the user can select a weight for each feature to customize.

$$Dissimilarity(VF_1, VF_2) = \sqrt{\sum_{i=i}^{F} W_i (VF_1 - VF_2)^2} \tag{3.1}$$

In the formula, VF_1 e VF_2 are the features vectors of the time series, F is the number of features, and W_i represents the weight of the features analyzed.

3.2 Clustering

In the **Shape-Based** approaches, many algorithms are based on the kMeans algorithm. The most powerful algorithm of this category is kShape[98]. kShape is a partitional clustering method that creates the prototypes of the k clusters by detecting and extracting the most peculiar shapes of each cluster. In brief, kShape works in two steps: in the refinement step, the time series prototypes are created by extracting the shapes that most reflect the changes in clusters; In the assignment step, the algorithm updates the cluster memberships by comparing each time series with all the computed prototypes and assigning each time series to the cluster of the closest prototype.

The other approach used in partitional clustering is kMedoid. [12] is a kMedoid algorithms that merges the PAM solution with a density-based [23] approach for obtaining the clusters. In the first step, the algorithm computes all the distances between the time series by adopting the DTW, finding series with many close neighbors. A series is considered close when the distance is lower than a specified cutoff distance d_c. Then series that lie in dense areas (i.e., that have lots of neighbors) are taken as centroids.

In the Semi-Supervised approaches for clustering the time series based on the shape, CobrasTS[136] is probably the most prominent. It creates the two kind of connection explained in Section 2.3.2 by asking the user if some time series could be placed in the same cluster or not. This operation, also called active learning, permits the creation of connections based on the user's knowledge.

The grouping of time series through interpretable **Features-Based** approaches is still in its early stages. The first work was proposed by [142]. The authors use a small set of measures for clustering the time series. Once extracted the features, the framework adopts a Hierarchical clustering algorithm for obtaining the group. [111] proposed the features-based time series clustering for analyzing the customer's electricity consumption and estimating the electricity network's load estimation. These pieces of information help discover the hours where the consumers use more electricity and optimize the load of the entire net. The method proposes to extract seven features from each customer's data. The features extracted were: mean, standard deviation, skewness, kurtosis, chaos, energy, and periodicity. Finally, the algorithms apply the kMeans for obtaining the group of customers with the same electrical load.

Another approach called Two-Stage Statistical Segmentation Clustering procedure (TS3C) [42] extracts the features not on the entire time series but its subsequences. The algorithm is divided into two main stages. In the first stage, the algorithm extracts the most interesting subsequences by adopting an algorithm called SwiftSeg. SwiftSeg creates a sequence that iteratively grows up and simultaneously is computed the corresponding least-squares polynomial approximation of the sequence and its error. The sequence grows until the error exceeds the threshold. When this occurs, the obtained sequence of points is saved, and the operation is continued until the time series is completed. After the SwiftSeg process, the sequences are mapped in arrays, including the polynomial coefficients of the least-squares approximation of the sequence and the set of statistical features extracted on each sequence. Finally, a clustering algorithm is applied to group all the time series sequences. The purpose is to represent each time series with the arrays and significantly reduce the representation size. The second stage of the method proposed consists of mapping the time series to a common representation. The idea is to represent each time series by the arrays created previously. Then, the centroid array, i.e., the average of all cluster points, and the sequence with higher variance are chosen. Finally, The time series having the most representative arrays in common are clustered in the same group.

3.3 Classification

The **Shape-Based** approaches are the most used methods for classifying the time series. The most used algorithm is the Learned pattern similarity (LPS)[10] algorithm. LPS creates an internal regression model capable of detecting correlations between subseries. LPS selects random subseries, and for each location, the subseries in the original data are concatenated to form a new attribute. The internal model selects a random attribute and constructs a regression tree. A collection of regression trees are processed to create a new collection of instances based on the counts of the number of subseries at each leaf node of each tree. Finally, the procedure for classifying a new time series is based on adopting a 1-NN classification of the concatenated leaf node counts. The LPS belongs to the interval-based classifiers.

Bag of patterns (BOP)[80] is a dictionary-based classifiers and applies SAX (presented in Section 2.3.1) to each sliding window of fixed dimension to form the words. This process will be repeated to evaluate a new unlabeled time series. Indeed, once obtained the new histogram, the class will be chosen by applying the k-NN algorithm, adopting the difference between the histograms as a distance. Another algorithm for creating words adopting windows is Bag of SFA Symbols (BOSS)[120]. The main difference between BOSS and BOP is the extraction of the words. Word features for BOSS classifiers are extracted from series using the Symbolic Fourier Approximation (SFA)[121]. This algorithm is essentially composed of two main steps. In the first step, the Fourier transformation is applied among all the sliding windows of the series. Then, the first l Fouriers terms will be discretized by applying the Multiple Coefficient Binning (MCB). This algorithm permits the division of the time series into a sequence of letters. Subsequently, the BOSS algorithm is equal to BOP.

In the family of the shapelet-based classifiers, [148] is the first shapelet classification algorithm that finds shapelets by enumerating all possible candidates, then uses the best shapelet as the splitting criterion at each node of a decision tree. In the last years, many shapelets-based algorithms have been proposed. For example, in [51] the authors once extracted all the possible shapelets candidates, adopt the k-NN for finding the shapelets with the smallest distances. This solution is poorly performed in terms of time. Therefore, the most recent version [16] balances the number of shapelets per class and evaluates each shapelet on how well it discriminates against just one class. The process is based on finding the k/c shapelets for each class, where c is the number of classes. The algorithm extracts all the possible candidate shapelets, computing the discriminatory power of each shapelet belonging to a class. The discriminatory power is computed by flowing the shapelets among the original series of the same classes. Then, the Euclidean Distance is computed between the shapelets and each subseries of the original series. This operation permits measuring how

well the shapelet discriminates the series (using the information gain) of a specific class, and the top k/c shapelets are retained.

Finally, in a new method called Ensemble-based approach, most of the previously presented solutions are merged to improve the quality of results. Collection of transformation ensembles (COTE)[5] is probably the most performed algorithm for classifying the time series. This algorithm merges different classifiers (and domains) in order to improve the accuracy of time-series classification. The algorithm combines classifiers in the time, autocorrelation, power spectrum, and shapelet domains. Any constraints are used for choosing the classifiers from each domain except for the time domain that are used a set of elastic similarity measures, such as all the derivatives of DTW. To achieve its high accuracy, COTE becomes hugely computationally intensive and impractical to run on a real big data mining problems.

Among the Neural Network solutions adopted for classifying the time series based on shape, the most straightforward approach was proposed by [144]. The network comprises three convolutional layers, each one performing a non-linear transformation of the input time series. Before the final classifier, a global average pooling technique is used. This final architecture is independent of the input time series length. Another interesting network is called Multi-Channels Deep Convolutional Neural Networks(MCDCNN)[60]. This network is composed of several couples of convolutional layers and max-pooling operations. Then, an MLP is used for classifying the series. A similar approach called Time Convolutional Neural Network (Time-CNN)[150] modifies the MCDCNN by adopting the mean squared error (MSE) instead of the traditional categorical cross-entropy loss function, which all the deep learning methods have used.

In the **Feature-Based** approach for classifying the time series, the first method is [94]. This method aims to extract some statistical features (Skewness, Mean, Standard Deviation, and Kurtosis). Each time series is represented by an array describing the value of the features, and these vectors will be the input of an MLP Neural Network. The results obtained by this method were already exciting, also with a small set of features.

Another algorithm that has adopted the features for classification is [147]. This method performs an early classification, i.e., it tries to optimize earliness under a requirement of minimum accuracy instead of optimizing accuracy in general classification methods. This method is the first approach that proposes a sort of selection of the most discriminant features among the ones extracted. It is divided into two stages: in the first stage, an extraction of local shapelets is applied to obtain the most interesting shapelets from the time series. For each local shapelet, a features extraction method is applied to detect the most distinctive features for time series classification. In the second stage, the algorithm selects a small subset of local

shapelets by the criteria of earliness, frequency, and distinctiveness. This step is beneficial for extracting features that are fruitful for early classification and opt-out of overfitting.

TSCLAS[15] is a new method for classifying the time series based on their main characteristics. This approach was proposed for classifying time series in the IoT domain, where the traditional classification algorithms are inapplicable. TSCLAS transforms the time series into the ordinal patterns domain, where it appears straightforward to capture the temporal dynamics of raw data. The process of transforming a time series into the ordinal pattern domains is based on two parameters: an embedding dimension D and an embedding delay τ. The parameter D permits the construction of sliding windows of size D, which will define the ordinal patterns. At the same time, the parameter τ corresponds to the timescale interval used to sample the consecutive points of those sliding windows. The importance of choosing the best parameters is evident. TSCLAS estimates the parameter by maximizing the class separability of the time series dynamics within the training dataset. Indeed, they consider the training dataset and its classification potential given the separation of their classes for different parameters. Finally, TSCLAS classifies the IoT data based on a combination of different features extracted from ordinal patterns probability distribution and the ordinal patterns transition graph, both derived transformations from the set of computed ordinal patterns. The features extracted from the ordinal distribution are similar to those extracted from the time series. Indeed, they adopt the normalized permutation entropy, the statistical complexity, and the Fisher information measure.

3.4 Anomaly Detection

In the actual state of the art, all the solutions proposed for detecting anomaly in time series are based on the **Shape-Based** approaches. The straightforward approach, called Subsequence Time-Series Clustering (STSC)[57], adopts KMeans for detecting anomalies. This algorithm divides the entire time series into *s* different subsequences. The user defines the number *k* of clusters required, and the kMeans process the entire dataset, providing the k final centroid. The anomalies are detected by computing the distances between the centroid subsequence and each subsequence of the cluster. The subsequences with a distance higher than a specified threshold are considered anomalies.

Other interesting anomaly detection algorithms are based on the Local Outlier Factor (LOF) [20]. LOF is an algorithm based on the kNN that computes the local density deviation of a data point, comparing it with the local density deviation of its neighbors. A data point is considered an anomaly if the local density of the data points is substantially lower than their neighbors. [97], as the previous algorithm, divide the time series into *s* different subsequences.

The obtained subsequences are subsequently divided into two datasets called ‘Training’ and ‘Testing’. The LOF on all the subsequences belonging to the Testing dataset is computed. The subsequences with a low value of LOF are considered anomalies.

The ability of the deep neural network to discover dependencies between the data fits very well for detecting anomalies in time series data. A simple method for detecting anomalies in time series is to adopt the MLP introduced in Section 2.3.3. During the process, a sliding window of length w flows on the time series, providing in input to the MLP model the subsequence. After a few iterations, the MLP model converges, and it can produce predictions on future subsequences. Therefore, these predictions will be used to compare with the observed subsequences. If the difference is huge, the observed subsequences will be evaluated as anomalous. Other deep neural network approaches adopt CNN for detecting anomalies. DeepAnT[91] proposes an approach similar to the one explained for the MLP but with a CNN network.

Long Short Term Memory (LSTM)[52] network is a Neural Network designed for the sequence data. It belongs to the Recurrent Neural Network(RNN) family, a family of Neural Networks where the data are treated sequentially. The difference between the feed-forward neural networks and the RNN is in the computation of the data. The RNNs work with sequence data; thus, the neurons’ weights are updated each time a new sequence is received. Therefore, the weights are already updated with the previous sequence when a new sequence occurs. This kind of Neural Network permits training the model incrementally without using the entire training dataset. [27] applies the LSTM for predicting the values of the given time series. It uses the probability distribution of the prediction error for marking timestamps as normal or anomalous.

Chapter 4

Time Series Features Clustering

4.1 Introduction

Time is a dimension that affects many aspects of real-world and digital-world phenomena. Physical environment, industrial machinery, healthcare monitoring, and economic and financial activities are a few applications whose components are regulated and evolve over time. In many of these applications, the time series dataset are widely used data artifacts for encoding collections of sequential observations over the temporal axis.

Time Series analytics includes supervised and unsupervised tasks, ranging from classification and clustering to pattern discovery, forecasting, and exploration. Cluster analysis recently gained momentum in many applications and use cases where sensors collect massive data points. Clustering aims to organize unlabeled data objects into homogeneous groups while minimizing intra-cluster dissimilarity and maximizing inter-cluster dissimilarity. Most of the solutions proposed in the literature are based on adopting raw data for computing the similarity between the series. However, these solutions are very sensitive to the noise in the data, making it impossible to apply these solutions in the real-life domain. The adoption of the features permits avoiding this problem. Indeed, the features permit a summarization of the series' principal characteristics and diminish the sensibility to the noise. Moreover, most of the tools proposed for extracting features allow to obtain fully explainable features. These tools lead to more interpretable analyses. An essential improvement in the time series analysis comes from studying the global relationship between the series. In the process of time series clustering, only the local relationships among neighbor data samples are identified, while global long-distance relationships remain unknown in general. However, in some time series analyses, capturing the global knowledge of the pattern formation of a given time series permits an improvement of the studies. Networks are a powerful mechanism for recognizing the long relationship between any pair or group of time series.

Research on Multivariate Time Series (MTS), i.e., datasets with more than one time-dependent variable (also called signal), is still at an early stage. Several proposals adopt clustering approaches designed for Univariate time series to Multivariate time series after applying dimensionality reduction techniques. The main problem with the Multivariate Time Series analysis is managing the high dimensionality of the data. In particular, in the MTS clustering analysis, most of the solutions aim to reduce the high dimensionality of the datasets. An example of such techniques is the Principal Component Analysis (PCA) [76, 75], which enables the conversion of a set of correlated features in the high dimensional space into a set of uncorrelated features in the low dimensional space. Nevertheless, the resulting clusters suffer from poor explainability as the original dimensions are lost. More recently, approaches based on Deep Neural Networks (DNNs) have been used to generate MTS encodings to apply clustering methods. Although these solutions might exhibit high performance, the resulting clusters are based on latent dimensions that remain unexplainable to the end-users. Explainability and interpretability are essential aspects of data science pipelines [114, 92], aiming at motivating the insights extracted from raw data.

This chapter introduces two novel techniques, called FeatTS and Time2Feat, for clustering respectively Univariate and Multivariate time series by adopting interpretable solutions and leveraging the concept of global relationships.

FeatTS algorithm revolves around feature-based clustering for univariate time series by providing a unifying framework in which global and local relationships between different time series and characteristic features are leveraged. Time2Feat is an extension to multivariate signals, in which intra-signal and inter-signal features are exploited and shown to be more effective and efficient than existing clustering approaches.

4.2 FeatTS: Clustering Univariate Time Series

In this section, we present FeatTS, a Semi-Supervised Clustering method that leverages features extracted from the raw time series to create clusters that reflect, as much as possible, the original time series. Our approach differs from existing methods in the literature since it employs the time series features, whereas existing methods focus on the similarity of the time series themselves. The novelty of FeatTS consists in automatically selecting the most appropriate statistical features based on the dataset provided as input, the latter characteristic being relevant for data science and data analytics pipelines. In fact, not all the features have the same quality, and choosing a subset of high-quality features for each dataset is beneficial for the clustering step. Moreover, the time series features are interpretable by humans, thus leading to a more transparent and human-centric clustering process. To the best

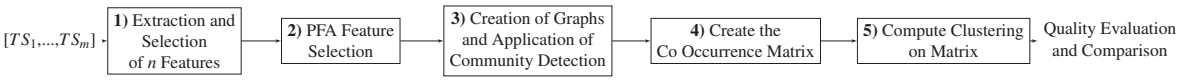


Fig. 4.1 The algorithmic pipeline of FeatTS.

a

Time Series	*mean*	*trend_stderr*	*variance*	*peaks*	*quantile*	*trend_rvalue*	Length	Label
TS_1	51.3	3.51	788.56	8	57	-0.94	89	No Kidney Failure
TS_2	40.6	4	128.9	5	43	-0.55	206	~~No Kidney Failure~~
TS_3	74.3	17	296.8	10	106	0.01	159	Kidney Failure
TS_4	95.8	9.4	783.3	10	85	0.43	139	~~Kidney Failure~~

quantile
trend_stderr
trend_rvalue

b

Dataset	TS_1	TS_2	TS_3	TS_4
TS_1	1	$\frac{1+0.5}{0.66+1+0.5}$	$\frac{0.5}{0.66+1+0.5}$	$\frac{0.5}{0.66+1+0.5}$
TS_2	$\frac{1+0.5}{0.66+1+0.5}$	1	$\frac{0.5}{0.66+1+0.5}$	$\frac{0.5}{0.66+1+0.5}$
TS_3	$\frac{0.5}{0.66+1+0.5}$	$\frac{0.5}{0.66+1+0.5}$	1	$\frac{0.66+1+0.5}{0.66+1+0.5}$
TS_4	$\frac{0.5}{0.66+1+0.5}$	$\frac{0.5}{0.66+1+0.5}$	$\frac{0.66+1+0.5}{0.66+1+0.5}$	1

c

Dataset	TS_1	TS_2	TS_3	TS_4
TS_1	1	0.69	0.23	0.23
TS_2	0.69	1	0.23	0.23
TS_3	0.23	0.23	1	1
TS_4	0.23	0.23	1	1

d

Fig. 4.2 A running example on real-world healthcare data.

of our knowledge, FeatTS is the first feature-based semi-supervised clustering framework with these characteristics.

4.2.1 Pipeline of FeatTS

The pipeline of FeatTS, as illustrated in Figure 4.1, is a combination of steps that contribute to the quality of the clustering results. We describe these steps in detail in the following.

4.2.1.1 Feature Extraction and Selection

The first step of the pipeline is the feature extraction step from the time series corresponding to step 1) in Figure 4.1. We consider feature extraction methods available in the literature and in readily predefined libraries [30]. Formally, given a vector of features $[f_1, f_2, \ldots, f_n]$ extracted, we construct a table containing the value of each feature, thus having as columns the features and having the time series $[TS_1, TS_2, \ldots, TS_m]$ as rows. As a simple example, in Figure 4.2a, we show an instance of the table for 4 time series and 6 features. Moreover, each time series is displayed with its corresponding label. The features shown in Figure 4.2a represent only a tiny subset of the set of features that can be extrapolated from the time series. Indeed, the tsfresh[30] library allows us to extract a significantly higher number of features. Therefore, *feature selection* becomes pivotal in our setting since not all the features have the same relevance for the subsequent clustering steps. In particular, we compute the relevance of the extracted features by solely using the feature values corresponding to the class label of the time series (e.g., in Figure 4.2a the class 'Kidney Failure' or 'No Kidney

Failure'). The Benjamin–Hochberg is a supervised procedure [13] that allows us to identify the relevance of the features based on the label associated with the time series. It computes the p-value of each feature provided as input based on their relevance. The p-value is a critical metric that allows us to quantify the significance of each feature. The Benjamin–Hochberg procedure output is a list of features ranked by p-values. Among these features, only a subset of them usually has an acceptable relevance. Indeed, a high p-value implies poor quality of the features. From our empirical study, it has been evinced that the top-20 features in order of relevance are sufficient to obtain high-quality clustering. Thus, we drop all the other features keeping the best 20. Usually, one of the main problems when computing the p-value is the redundancy of the obtained features. Finding a duplicate-free combination of the features is desirable, preserving the quality and little number. Therefore, once we select the 20 features from the list produced by Benjamin–Hochberg, we need an algorithm that allows us to find a minimum number of features representative of the other features not included in the analysis. Dimensionality reduction comes at hand to help us reduce the dimensionality of the feature space by obtaining a set of principal features. We apply a technique called Principal Feature Analysis (PFA) [82]. We fix a value t of the explained variance in our experiments equal to 0.9. This value is the best result empirically produced with various threshold values t. In our example, among all the features presented in Figure 4.2a, we have selected only $quantile$, $trend\_stderr$ and $trend\_rvalue$, as shown in Figure 4.2b. In our experimental analysis, we obtained a different number (always less than 20) of features out of the PFA feature selection, the final number depending on the particular dataset.

4.2.1.2 Graph Rendering and Community Detection

We convert the time series and their relationships into edge-weighted graphs. Encoding time series into edge-weighted graphs allows us to represent our clustering problem in another dimension space without loss of information. This operation is crucial in order to be able to capture the global relationships among the raw time series samples.

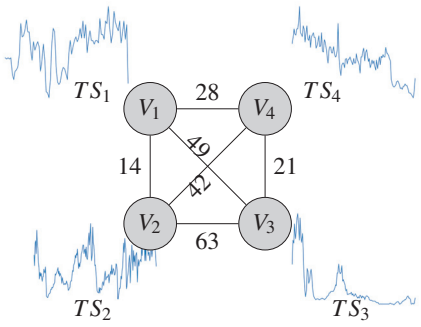
Suppose we have a feature F_i (as selected by PFA in the previous step) and a set of n time series $\{TS_1, .., TS_n\}$. Let TS_i be a node v_i in the set of vertices V of a graph G. Let E be the set of edges of graph G, where each edge e_i connects two nodes in G representing two different time series. Let $w : E \rightarrow \mathbb{R}$ be an edge-based weight function. Each edge e_i is thus assigned a weight $w(e_i)$ representing the distance between the connected nodes of the edge, i.e., the difference between the values of two different time series using the feature F_i. In order to capture similarity, we only retain in G the edges whose weight is less than a given threshold distance th.

Example 1. *As an example, let us consider the four time series as in Figure 4.2a, each of which has the values of $quantile$; we will compute all the distances between these values. Figure 4.3a shows the graph encoding of these time series where the weights on the edges represent the distance between the time series, based on $quantile$.*

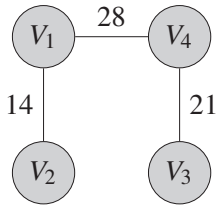
One immediate question is the choice of the threshold th. Given n nodes in G corresponding to the n time series, there are $\frac{N*(N-1)}{2}$ distances between all pairs of nodes. To capture similarity, we use a simple heuristic of a percentage x representing the proportion of the smallest distances to be kept. The threshold th is thus selected based on this x percentage.

Example 2. *For instance, for the graph in Figure 4.3a, the array in Figure 4.3b contains the distances between the vertices in ascending order. Suppose that the user specifies 50% of the vector as the percentage. This amount implies that the distance boundary will be* 28*, and the distances higher than* 28 *will be discarded (i.e., the corresponding edges in the graph will be ignored). Once we have chosen the boundary distance, we can create the corresponding graph as depicted in Figure 4.3b.*

Notice that a higher threshold would consider lower significance edges and thus weaker similarities between the times series. On the other hand, a lower threshold may cut important edges. In our empirical evaluation, we used a threshold determined by a user-specified percentage of 80%, which works well in practical scenarios, as we will see in the remainder of the section. The chosen threshold will be used for all features selected by PFA and thus for all graphs created. Each graph is created based on one selected feature from PFA in the previous step. Thus, if PFA selects k features, there will be k graphs, each corresponding to one notion of similarity between a pair of time series. The intention is to combine these different notions of similarity in time series clustering, by leveraging the structures of connectivity in the various graphs. To this purpose, we apply a community detection (CD) algorithm in order to search for groups of densely connected vertices forming communities. Among the different tested algorithms, we have opted for the Greedy Modularity Algorithm [95] in the NetworkX library [45]. This algorithm turns out to strike a balance between speed and robustness and does not require any additional input parameter other than the graphs. In Figure 4.4, we show an example of clustering obtained by applying this algorithm to a family of graphs. We can notice that the clustering varies from one graph to another graph. A natural question is how we can unify the different clusters in order to obtain understandable results.



(a) Edge-weighted graph with distances as weights.



V_1	V_2	**14**
V_3	V_4	**21**
V_1	V_4	**28**
—	—	—
V_2	V_3	42
V_1	V_3	49
V_2	V_4	63

(b) The graph for a single feature after application of the threshold

Fig. 4.3 Encoding from time series to graph.

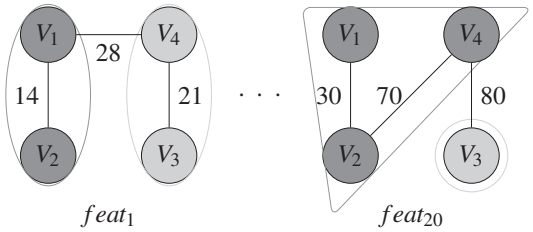


Fig. 4.4 Application of Community Detection algorithm for each feature.

4.2.1.3 Co-Occurrence Matrix

The underlying intuition is that if two time series are similar, they will be similar for the majority of their discriminating features. We employ a *co-occurrence matrix* [93] to put this in practice. The matrix records each pair of time series and how many times they are grouped within the same community. Intuitively, the more they are placed within the same community, the more similar the time series are.

Co-Occurrence Matrices without weights. Assuming we have M time series and L features, we know that, once applied the CD algorithm on the L graphs, we will obtain the following result:

$$Feature_1 = \{(TS_1, TS_3, ..., TS_s), ..., (TS_2, TS_4, ..., TS_p)\}$$
$$Feature_2 = \{(TS_2, TS_3, ..., TS_t), ..., (TS_1, TS_4, ..., TS_i)\}$$
$$...$$
$$Feature_n = \{(TS_2, TS_1, ..., TS_m), ..., (TS_3, TS_4, ..., TS_q)\}$$

where for each feature $Feature_i$ selected by the PFA, we obtain different communities $(TS_1, ..., TS_i)$ composed by the time series. We can now create a matrix in which the rows and columns contain all the time series of the dataset. Each cell x_{ij} in the matrix corresponds

to the similarity between time series TS_i (in row i of the matrix) and TS_j (in column j of the matrix).

Next, we convert the counts in the Co-Occurrence matrix into a similarity metric for the eventual clustering. We consider the number of times that a pair of time series is present in all possible communities where at least one of the two time series belongs. That is, given the time series TS_i, TS_j, the communities C and the set of all the time series M, the similarity between TS_i and TS_j will be as follows.

$$\forall\ TS_i, TS_j\ \in M,\ \forall c \in C\ x_{ij} = \frac{\left|\{\ c \in C \mid TS_i \in c\ \ \&\ \ TS_j \in c\ \}\right|}{|\{c \in C|\ TS_i \in c\}|} \tag{4.1}$$

That is, the number of times that the two time series TS_i and TS_j fall within the same community divided by the number of times that TS_i is found within any community. Notice that (1) is entirely symmetrical. Indeed, the communities found for each feature are considered hard clustered, i.e., a time series TS_i cannot be part of two communities of the same feature and must necessarily belong to one. Thus, if TS_i and TS_j are within the same community of a specific feature, neither of them can be part of other communities of the same feature. Therefore, the value x_{ij}, given by the number of times TS_i and TS_j are in the same community, will be equal to x_{ji} because TS_j and TS_i must also be in the same community.

Co-Occurrence Matrices with Weights. Applying the CD algorithm and its processing with co-occurrence matrices without weights might incur the problem of community fragmentation. More precisely, the CD algorithm might lead to the formation of many communities, each containing a few time series. This effect is because some features are often not discriminatory enough for that dataset. To overcome this problem, we assign an approximate weight to each feature based on the number of communities that the CD algorithm derives from the graph. Again, to correctly determine the weights, we require the user's input on the expected number of clusters.

Let w_i be a weighting function defined on each feature F_i as follows:

$$\begin{cases} w_i = \frac{C}{O_i},\ if\ O_i > C \\ w_i = \frac{O_i}{C},\ if\ C > O_i \\ w_i = 1,\ otherwise \end{cases} \tag{4.2}$$

Where C is the number of clusters expected by the user and O_i is the number of communities extracted through the CD algorithm. Hence, the weights will be higher if the number of obtained communities O is equal or sufficiently close to C and lower otherwise. The weights will be propagated to the similarity matrix, which will now reflect the importance of each feature from a user viewpoint. Not surprisingly, instead of simply counting the times that the

time series TS_i and TS_j co-occur in the same community, we sum their weights and divide by the sum of the weights of the all-time series, as also shown in the following.

Example 3. *As shown in Figure 4.2b, the PFA feature selection has chosen three features, namely* $[trend\_stderr, quantile, trend\_rvalue]$*. After applying the CD algorithm, we obtained the following communities (per feature) among the* 4 *Time Series in Figure 4.2a:*

$$quantile = \{(TS_1, TS_2), (TS_3, TS_4)\}$$
$$trend\_stderr = \{(TS_1), (TS_2), (TS_3, TS_4)\}$$
$$trend\_rvalue = \{(TS_1, TS_2, TS_3, TS_4)\}$$

Assume that the user specifies an expected number of clusters equal to 2*.* $Trend\_stderr$ *and* $trend\_rvalue$ *do not satisfy the number of clusters expected by the user. Therefore,* $trend\_stderr$ *will have a weight of* $\frac{2}{3}$ *(*0.66*), while* $trend\_rvalue$ *will have a weight of* $\frac{1}{2}$ *(*0.5*), and* $quantile$ *we will have* 1 *as weight. We report the intermediate computation of the co-occurrence matrix with weights for this example in the Table in Figure 4.2c and the final result in the Table in Figure 4.2d.*

4.2.1.4 Clustering the Co-Occurrence Matrix

The co-occurrence matrix obtained in the previous step allows us to quantify the similarity between two time series. In order to be prepared for the creation of the time series clusters, we need one more intermediate step, i.e., to compute the distances between the rows of the Co-Occurrence Matrix. We employ a standard Euclidean distance to perform the row comparison.

As an example, applying the Euclidean distance between the rows of the table in Figure 4.2d, we obtain Table 4.1. For instance, the value of the cell $C_{3,4}$ of the Table 4.1 is 0 because the row 3 and 4 of the Table in Figure4.2d are equal. Consequently, the time series 3 and 4 are always together in each cluster discovered by the CD algorithm. Finally, we apply the standard k-Medoid algorithm [100] on the distances computed above. K-Medoid allows us to extract the time series with the smallest distance.

To complete our running example, applying the k-Medoid algorithm to Table 4.1 requiring 2 clusters as the input parameter, we obtain two clusters $Cl_1 = \{TS_1, TS_2\}$ and $Cl_2 = \{TS_3, TS_4\}$, as shown in the Table in Figure 4.2d.

The time complexity of the FeatTS is summarized in Lemma 1.

Lemma 1. *Let D a dataset composed by m time series and let L the number of features chosen by PFA among the N features extracted and k the number of requested clusters. A*

Dataset	TS_1	TS_2	TS_3	TS_4
TS_1	0	0.64	1.36	1.36
TS_2	0.64	0	1.36	1.36
TS_3	1.36	1.36	0	0
TS_4	1.36	1.36	0	0

Table 4.1 Co-Occurrence Matrix with weights.

dataset D is evaluated by FeatTS in time $O(L(m^2)+m^2+k(m-k)^2+n\cdot t_f)$ *and in space* $O(n+L(m+E)+m^2)$.

Proof: Let D a dataset composed by m time series and let L the number of features choose by PFA among the n extracted. A dataset D is evaluated by FeatTS in time $\Omega(L(m^2)+m^2+k(mk)^2+n\cdot tf)$ and in space $\Omega(n+L(m+E)+m^2)$.

Time Complexity. Starting from the Community Detection, the Greedy Modularity algorithm requires $\Omega(m^2)$ time to extract the community from a graph with m nodes. FeatTS transforms each time series in a node; thus in a dataset of m time series, one community is detected in time $\Omega(m^2)$. However, this cost is related to a single feature. Indeed, FeatTSFeatTS creates as many graphs as the number of features chosen by PFA. Hence, assuming PFA chooses L features, the total cost will be $\Omega(L(m^2))$. The time complexity required by the Co-Occurrence Matrix depends strongly on the computation of Equation (1.1). Indeed, since we have to estimate the number of times in which two time series fall within the same community, we can easily deduce that creating the Co-Occurrence Matrix will take time $\Omega(m^2)$. Finally, the time requested by the kMedoid to extract the cluster in the Co-Occurrence matrix is equal to $\Omega(k(mk)^2)$, where k is the number of requested clusters. A further component used in the pipeline is features extraction. TSFresh does not provide the time complexity of the method proposed but the time complexity required will be equal to the time needed by each feature to be extracted. Therefore, fixing as tf the average extraction time of a single feature, the time required to extract the features is given by $n\cdot tf$, where n is all the extracted features.

Space Complexity. We evaluate the space required by each component of the pipeline. In the first step, TSFresh allows us to extract plenty of features, where each feature occupies constant memory. Therefore, suppose n are the features extracted by TSFresh, the features extraction component requires $\Omega(n)$ space in memory. In the Community Detection step, each community is represented by a graph. Usually, a graph in memory occupies a space equal to $\Omega(V+E)$, where V are the vertices and E the edges. Therefore, suppose that m are all the time series, it is required $O(m+E)$ space in memory for each feature chosen by the PFA. Thus, suppose that L are all the features chosen by PFA. This Community Detection step occupies $\Omega(L(m+E))$ space in memory. Lastly, the space occupied by the

Co-Occurrence Matrix is equal to $O(m^2)$. The space required by the Cluster is constant and thus can be omitted.

4.2.2 Real-world Clinical Data

We use real-life time series courtesy of the Personalized Medicine Department at the European Hospital George Pompidou in Paris. These time series contain signals from patients suffering from kidney diseases. Kidneys have many functions, including maintaining acid-base balance, regulating fluid balance, regulating sodium, potassium, and other electrolytes, removing toxins, absorbing glucose, amino acids, and other small molecules, and regulating blood pressure, producing various hormones. Much of renal physiology is studied at the nephron level, the kidney's smallest functional unit. Each nephron begins with a filtration component that filters the blood entering the kidney. This filtrate then flows along the length of the nephron, a tubular structure lined by a single layer of specialized cells and surrounded by capillaries. The primary functions of these lining cells are the reabsorption of water and small molecules from the filtrate into the blood and the secretion of wastes from the blood into the urine. Proper function of the kidney requires that it receives and adequately filters blood. This function is performed at the microscopic level by many hundreds of thousands of filtration units called renal corpuscles, each composed of a glomerulus. A global assessment of renal function is often ascertained by estimating the filtration rate, called the glomerular filtration rate (GFR). GFR estimates how much blood passes through the glomeruli each minute. Glomeruli are the tiny filters in the kidneys that filter waste from the blood. A renal failure is detected when GFR is under $90mL/min/1.73m^2$. Whereas when it drops to $15mL/min/1.73m^2$, it means that the patient needs dialysis or a transplant. Thus, it is very important to understand when a patient needs medical treatment before the GFR reaches its lowest possible value. Moreover, since dialysis is an invasive operation, it is important to understand if a sudden drop in the GFR occurs. In this case, medical doctors might recommend urgent surgery or to resort to dialysis depending on the GFR values over time.

4.2.3 Experimental Setup and Results

We ran experiments on two variants of the Kidney dataset. The first variant, named $Kidney3Yr$, contains 222 patients (one time series per patient) and spans 1 to 3 years with a variable length between 90 and 230 data points in the time series. The second variant, called $Kidney5Yr$, is composed of 278 patients spanning five years, with the time series having roughly 100 data points. In both cases, we ran our experiments using only the 20% of the labeled time series to compute the set of features necessary to run the clustering algorithm

and to emulate the real-world scenario where not all the labels of the data points are available. The features thus being ordered based on their relevance have been employed to cluster the entire unlabeled dataset into those with GFR signals concerning high-risk patients and those containing GFR for patients with lower risk.

We also used 64 benchmark datasets from the UCR collection[33], including both real-life and synthetic datasets. The entire list of datasets used for a benchmark is available online[1]. For consistency, we used only 20% of the labeled time series during feature extraction during the clustering step in all experiments.

For each dataset, we consider 20 as the upper bound of the number of features we consider in the analysis. A higher number of features are supported by our method but are not indispensable for obtaining better accuracy. Moreover, a higher number of features deteriorates the performance. Furthermore, we chose 80% as the threshold value of the percentage of features selected by the user. We used the AMI [113] metric, a well-established measurement of the quality of clustering. We adopt the same metric for our comparisons as well.

We consider two main baselines, the first being the state-of-the-art algorithm for time series clustering, i.e., kShape[99], and the second being the state-of-the-art algorithm for Semi-Supervised time series clustering, i.e., Seeded kMeans [9], sharing the same category of our approach. kShape[99] could not be used on the real-world GFR time series as it cannot process variable-length time series. Hence, we limit the comparison with kShape to the UCR datasets. Other baselines of semi-supervised clustering algorithms such as SSSL[140] (Self Labeling Algorithm) and SUCCESS[85] (Cluster then Label Algorithm) could not be used in our study due to the lack of available source code. All experiments have been executed on a Linux server with 64GB of RAM, Intel Xeon CPU Skylake, and IBRS @ 2.6GHz.

4.2.3.1 UCR Dataset

The results in Table 4.2 are an excerpt of the entire results obtained by our algorithm and its baselines for various UCR datasets. It can be observed that FeatTS obtains the best results among all. Indeed, out of 64 datasets used for the comparison, FeatTS performed better on 37 datasets. In addition, kShape performed well only on 15 datasets (out of 64) and Seeded kMeans outperformed the others in only 12 datasets (out of 64).

[1]https://github.com/DonaTProject/FeatTS

Dataset	FeatTS	kShape	SeededKMeans
Adiac	0,31	0,39	**0,52**
MoteStrain	**0,48**	0,01	0,02
TwoLeadECG	**0,88**	0,10	0,07
ECG200	**0,34**	0,11	0,06
Computers	0,09	**0,06**	0,01
Coffee	**1**	0,35	0,88
GunPoint	**0,52**	0	0
Arrowhead	**0,29**	0,26	0,27
ItalyPowerDemand	**0,54**	0,39	0
Meat	0,4	0,64	**0,75**
OliveOil	0,27	0,52	**0,53**
Trace	**0,74**	0,52	0,69
Wine	**0,12**	0	0,01
Worms	**0,16**	0,06	0,12
ShapesAll	0,08	0,62	0,45

Table 4.2 Results showing the values of AMI for UCR datasets

Dataset	FeatTS	SeededKMeans
Kidney3Yr	**0.56**	0.44
Kidney5Yr	**0.58**	0.48

Table 4.3 Results on Kidney 3Yr and 5Yr Datasets

4.2.3.2 Kidney Dataset

As shown in Table 4.3, the results obtained by FeatTS are significantly more accurate than Seeded kMeans on the clinical case study. For the patients under medical supervision for 5 years, as shown in Table 4.3, we have obtained results following a similar trend.

4.2.4 Ablation and Scalability tests

In this section, we present a handful of ablation tests devoted to showing the importance of each step of the FeatTS pipeline. These tests were performed on several datasets belonging to the UCR. Moreover, we test the scalability of our system by adopting a time series generator. The tests are based on increasing the length and the number of the time series.

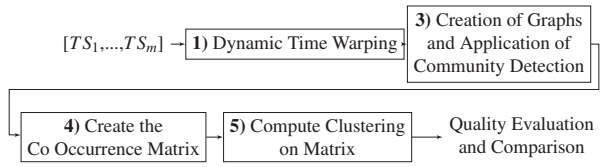


Fig. 4.5 DTW Pipeline

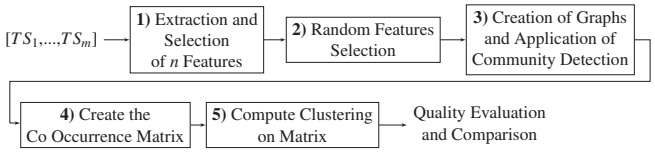


Fig. 4.6 Random Features Pipeline

4.2.4.1 DTW versus Features

The idea behind this ablation test is to show the importance of adopting the distance between the features instead of using the distance between the raw time series using Dynamic Time Warping (DTW) [14, 90]. In a nutshell, this corresponds to removing steps 1 and 2 from the pipeline in Figure 4.1, i.e., the extraction and the selection of the features using PFA, as shown in Figure 4.5. The Table 4.4 shows the results obtained by replacing the distances between the features with the distance computed by the DTW directly on the time series. The results are expressed in the column called *DTW*.

The results showed that 14 datasets (out of 15) have a worse behavior if DTW is adopted. Indeed, only in one dataset (*OliveOil*) does the usage of DTW outperform the features. On average, we have a difference in terms of AMI of 0.31. This ablation test confirms the importance of using distances between features instead of merely using distances between raw data.

4.2.4.2 p-value versus Random

Figure 4.6 shows the pipeline removing the ordering of the features based on their relevance, and replacing with a random features selection. We repeated this test 5 times, and we averaged the results. The column Rand represents the average results obtained by FeatTS using Random Features.

The results in Table 4.4 show that the algorithm's performance drastically deteriorates for most of the datasets (13 out of a total of 15 datasets) if random features are employed. Indeed, computing the average overall results of the two experiments, we have a difference of 0.35 in AMI. Therefore, ordering the features based on their relevance turns out to be indispensable for achieving good results.

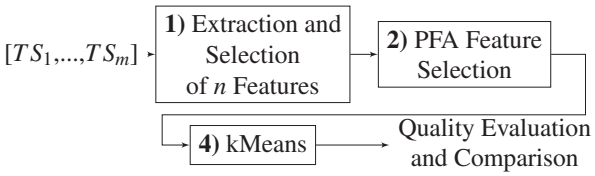


Fig. 4.7 kMeans Ablation Pipeline

4.2.4.3 kMeans versus Global Relation

This ablation test aims to show the importance of capturing the global relationship among the raw time series samples through graph encoding and of the subsequent application of the Community Detection algorithm as in this approach. Therefore, we replace steps 3, 4, and 5 as in Figure 4.1 with k-Means, i.e., a classical clustering algorithm in its multidimensional version. Hence, once the features have been extracted and selected through PFA in steps 1 and 2, we apply the k-Means algorithm to obtain the clustering, as shown in Figure 4.7.

In Table 4.4, we show the results capturing the global relationship among the raw time series samples through graph encoding and the subsequent application of the Community Detection. The column k-Means shows the results obtained by k-Means among the features selected by each dataset used in this experiment. We have highlighted in bold the best results obtained between k-Means and FeatTS for each dataset.

The results show that FeatTS outperforms kMeans in the majority of the cases. Indeed, there are only four datasets for which k-Means show slightly better results, namely UMD, $Meat$, $Coffee$, and $OliveOil$. On average, the results obtained by FeatTS outperform those obtained by k-Means of 0,08 in terms of AMI. Therefore, this ablation test shows the importance of capturing the global relationships between the various time series to achieve better performance results.

4.2.4.4 Scalability

We have assessed the scalability of our method by increasing both the number and length of time series in a dataset. In this experiment, we have used synthetic time series generated with GRATIS [63]. This tool allows a controlled generation of time series by using diverse characteristics such as spectral entropy, trend, seasonality, stability, etc. In the synthetic generation, we have opted for spectral entropy and trend as the underlying characteristics since they reflect the real-life time series we have used in the rest of our experimental assessment. The spectral entropy allows for measuring the "forecastability" of a time series. It has a range of values between 0 and 1, and a low entropy value indicates a high signal-to-noise ratio, while large values occur when a time series is challenging to forecast. For this

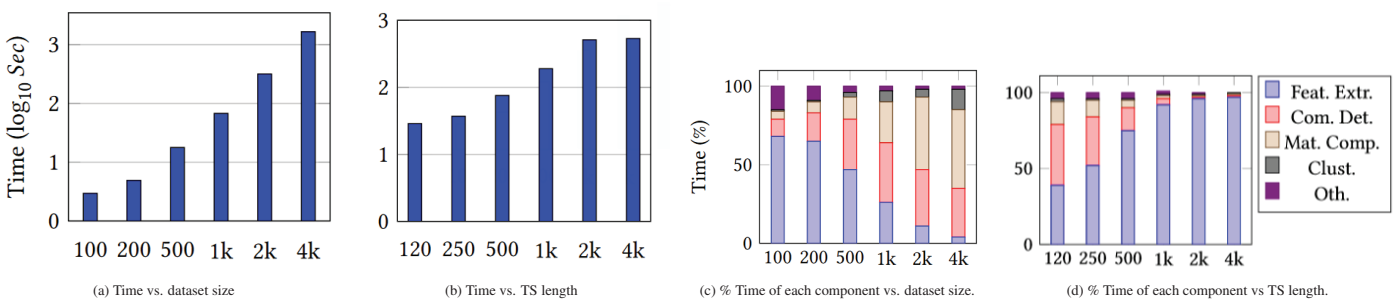
Dataset	FeatTS	DTW	Rand	kMeans
ScreenType	**0,02**	0,01	0	0,01
UMD	0,3	0,27	0,01	**0,42**
TwoLeadECG	**0,88**	0	0,02	0,75
ECG200	**0,32**	0,08	0,06	0,16
Computers	**0,09**	0	0	0
Coffee	0,8	0,01	0,12	**0,9**
GunPoint	**0,56**	0	0	0,26
Arrowhead	**0,28**	0,2	0,02	0,24
ItalyPowerDemand	**0,57**	0	0	0,51
Meat	0,42	0	0,16	**0,48**
OliveOil	0,15	0,32	0,2	**0,35**
Plaid	**0,35**	0,01	0,11	0,03
Asphalt Regularity	**0,54**	0	0	0,35
Asphalt Obstacles	**0,38**	0,27	0,02	0,37
Gesture Pebble	**0,25**	0,02	0	0,16

Table 4.4 Ablation Test Results.

characteristic, we have fixed a value of spectral entropy equal to 0.6. Conversely, the trend represents low-frequency variations in the time series. It has a range of values between 0 to 1, and we have chosen a value of 0.9 for this experiment.

In the first experiment, we increase the number of time series for each tested dataset while the length of the time series is fixed and equal to 60. Figure 4.8a shows the results obtained on datasets consisting of 100, 200, 500, 1000, 2000, 4000 time series, respectively. The results show the scalability of the method in terms of time performance, while a significant increase can be observed when shifting to more than 2000 time series. The times in Figure 4.8a are in logarithmic scale for clarity of exposition.

We have studied the percentage of time due to each pipeline component as shown in Figure 4.8c. Upon increasing the dataset size, the component that is computationally more demanding is the creation of the co-occurrence matrix. Obviously, since the Co-Occurrence Matrix depends on the number of time series, the time required for its creation increases as the size increases. In the second experiment, we increased the time series length while fixing to 500 the number of time series belonging to each dataset. Figure 4.8b shows the results obtained by increasing the time series length between 120 and 4000. The figure shows the scalability of the approach for time series under 2000 and a sudden increase of the time beyond this value. The time breakdown in Figure 4.8d shows that the more expensive step of the pipeline for this experiment is the feature extraction step.



(a) Time vs. dataset size (b) Time vs. TS length (c) % Time of each component vs. dataset size. (d) % Time of each component vs TS length.

Fig. 4.8 Scalability Results.

4.2.5 Human Centered Settings

In this section, we discuss the human-in-the-loop aspects of FeatTS by focusing on the parameters that can be changed along the pipeline. Figure 4.9 shows which component of the original pipeline the human can tune to influence the clustering process. The tuning takes place at the end of each step. The system allows the visualization of the output to appreciate the effect of the choice.

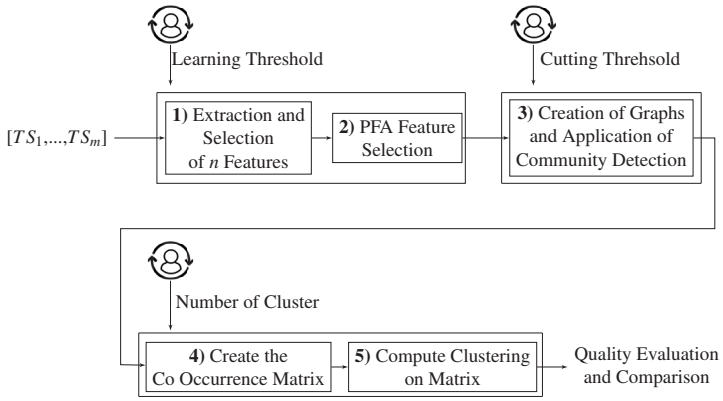


Fig. 4.9 The algorithmic pipeline of FeatTS.

4.2.5.1 Learning Threshold

The personalization of the features based on the user needs is the first problem faced by FeatTS. This parameter tunes the amount of supervision the system uses, increasing the relevance of the features that best fit with the labels provided as input. The choice of supervision depends on the user's purpose. The resulting clusters obtained with a high amount of supervision are strongly similar to the labels provided in input. This solution is fruitful when the user does not care about revealing others' information about the data. Conversely, when using a small or zero amount of supervising, the algorithm will favor the

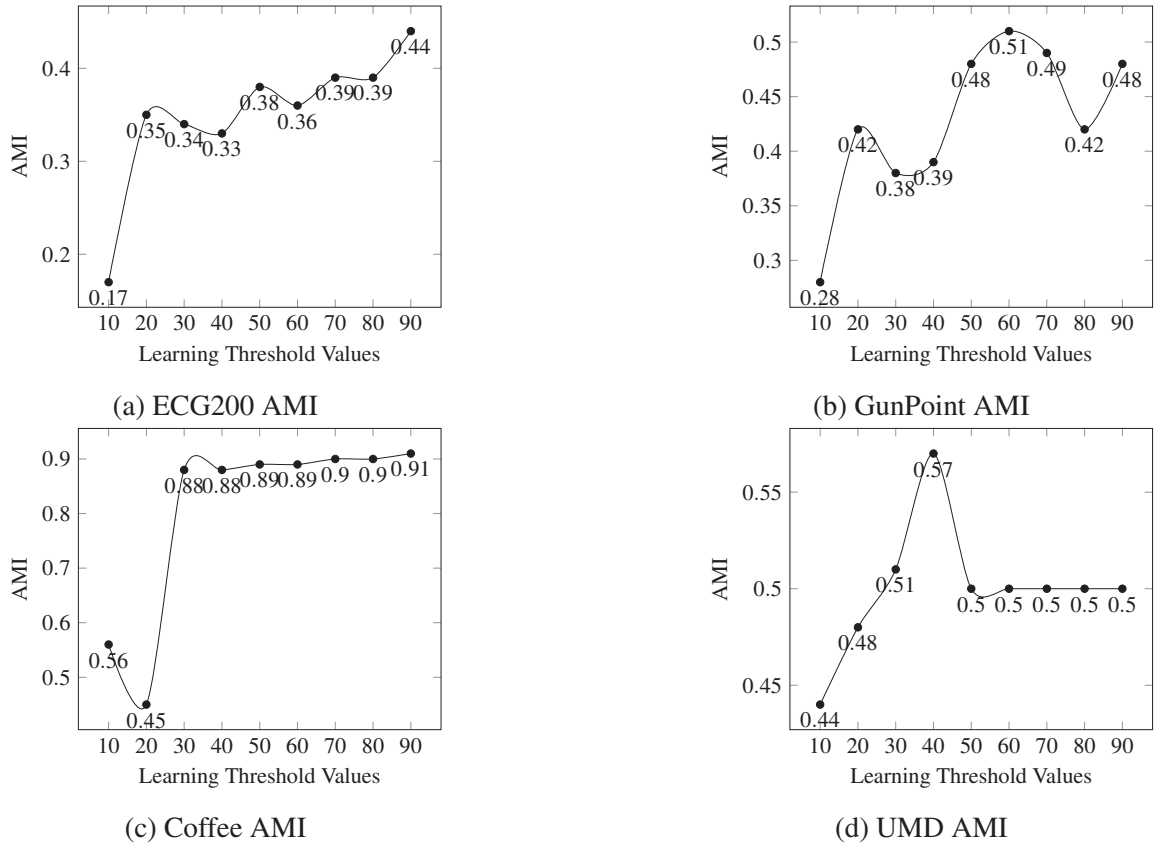
features based on their variance. In this case, the obtained clusters are less aligned with the input labels, but they can be better coupled with the raw data (i.e., the original time series). A parameter called 'Learning Threshold' will help the user choose the right amount of supervision. Figure 4.1 shows that the Learning Threshold parameter impacts steps 1 and 2 of the pipeline. In particular, the Benjamini-Yekutieli procedure will leverage the labels provided by the user during the supervised phase to provide the importance of each feature. Subsequently, the PFA leverages the features' importance to select the subset of features.

We evaluate how FeatTS behaves when the Learning Threshold changes. In this experiment, we consider the quality of the resulting clusters by comparing the original labels of the dataset with those obtained from the final clusters. The purpose is to evaluate how the quality of the resulting clusters increases as the learning threshold increases. This operation will be repeated for each value of the threshold. In our case, we decided to increase the threshold by 10% starting from 10% up to 90%. The quality of the clusters is measured with the Adjusted Mutual Information (AMI).

By modifying the amount of supervision and then increasing the Learning Threshold, we show how the quality of the clustering process can be improved. The experiments in Figure 4.10 show how the quality of the resulting clusters increases where the Learning Threshold increases. In some datasets, the same features can be selected at low and high threshold values. This result is often due to a small set of features that can be extracted. Indeed, as shown in 4.10d, this increase stops when we choose to use only 40% of the labels. This stop happens because, from 50% onwards, the features chosen by PFA are equal. Therefore the results obtained by FeatTS are the same.

4.2.5.2 Cutting Threshold

The primary goal of any clustering algorithm is to detect the similarity between the input data points. At this stage, we want to leave the users the possibility of playing with the quality of the similarity by using a parameter that allows them to increase or decrease the similarity between the various time series. FeatTS extracts and evaluates the global relationships between the time series by creating weighted edges. As Section 4.2.1.3 explains, the weights represent the distances computed by subtracting the absolute values of the feature of two connected time series. Hence, to obtain the level of similarity desired by the user, we propose the 'Cutting Threshold' parameter. This parameter, which represents the third step of the pipeline in Figure 4.9, indicates the percentage of lower distances to be kept for each feature encoded as a graph. Suppose that the user specifies a percentage of 50%, half of the edges will be cut. Theoretically, if the user wants to obtain small clusters formed by time series with a high similarity, they must delete several edges within the graphs. Hence, they have



(a) ECG200 AMI

(b) GunPoint AMI

(c) Coffee AMI

(d) UMD AMI

Fig. 4.10 Test Learning Threshold.

to choose a low value of the cutting threshold. On the other hand, if the aim is to obtain large clusters at the expense of similarity, in this case, the user should keep as many edges as possible. For this reason, a high cutting threshold is preferred. Once the edges are pruned, FeatTS extracts the actual global relationships through Community Detection. We evaluate how the similarity of the time series within community detection varies based on the value of the Cutting Threshold provided by the user. Theoretically, the similarity between time series should decrease as the percentage of the Cutting Threshold increase. In order to prove this statement, we extract all the communities from the features chosen by PFA. Subsequently, we compute the distances between the time series belonging to each community by leveraging DTW [14]. These distances will then be combined into an average for each community. This computation is shown in the Formula 4.3 where M_{c_k} represents the average of all the time series TS that belong to a single community C_k.

$$M_{c_k} = \forall\, TS_i, TS_j \;\in\; C_k,\; \frac{\sum_{i,j=0}^{|C_k|} DTW(TS_i, TS_j)}{\frac{|C_k|\cdot|C_k-1|}{2}} \tag{4.3}$$

The obtained average of the single community will be combined again with the average of the other communities for the features chosen by PFA.

Therefore, assuming that C is the set of all the communities found for all the features, we compute the final average as in formula 4.4. This final average called M_{th} corresponds to the distance average of the chosen threshold.

$$M_{th} = \forall\, C_k \in\ C,\ \frac{\sum_{k=0}^{|C|} M_{c_k}}{|C|} \tag{4.4}$$

In this experiment, we set a percentage of Learning Threshold of 20%, and the number of clusters equal to those required within the supervised dataset.

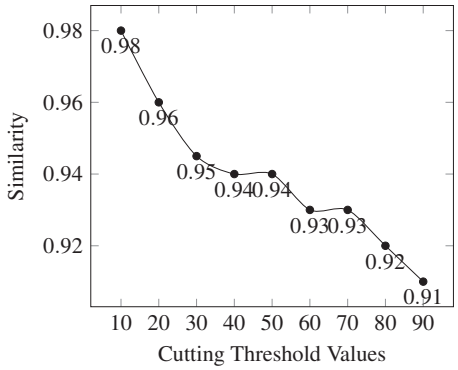
Table 4.11a shows that the average of the distances between the time series belonging to each community on the GFR Dataset increases as the Cutting Threshold increases. The increase of the threshold corresponds to the decrease of the similarity. Indeed, in Figure 4.11b, we can see that the drop of the similarity is about 7%. In order to compute the similarity, we have applied the Formula 4.5. Indeed, assuming that TS_i and TS_j are two time series that belong to the dataset, we have normalized the average distances of each threshold M_{th} with the maximum distance found between two time series

$$S_{th} = 1 - \frac{M_{th}}{max(DTW(TS_i, TS_j))} \tag{4.5}$$

In Figure 4.12 we report some results for UCR datasets that are compatible with those obtained with the GFR dataset and show a similar trend.

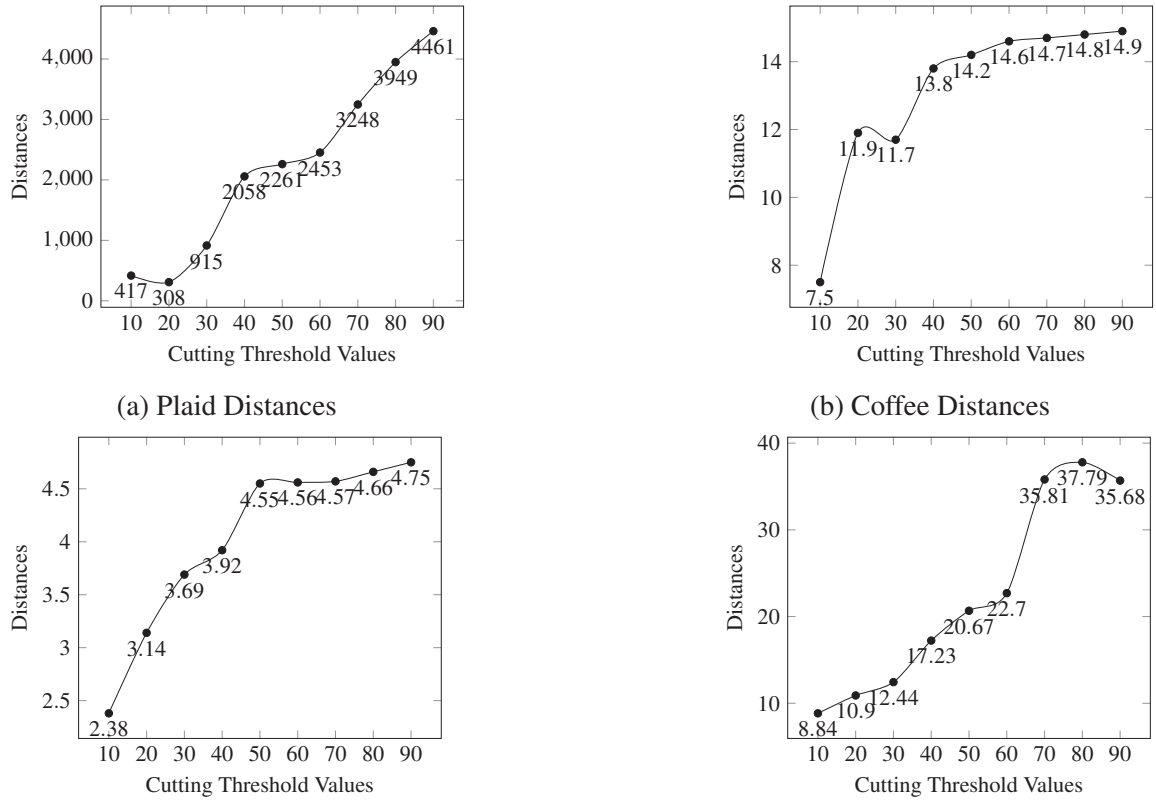
Threshold	Average Distances
10%	2449
20%	3060
30%	3966
40%	4366
50%	4354
60%	4625
70%	4648
80%	4661
90%	4662

(a) Average Distances



(b) Similarity

Fig. 4.11 Similarity and distances within the Communities.



(a) Plaid Distances

(b) Coffee Distances

(c) Meat Distances

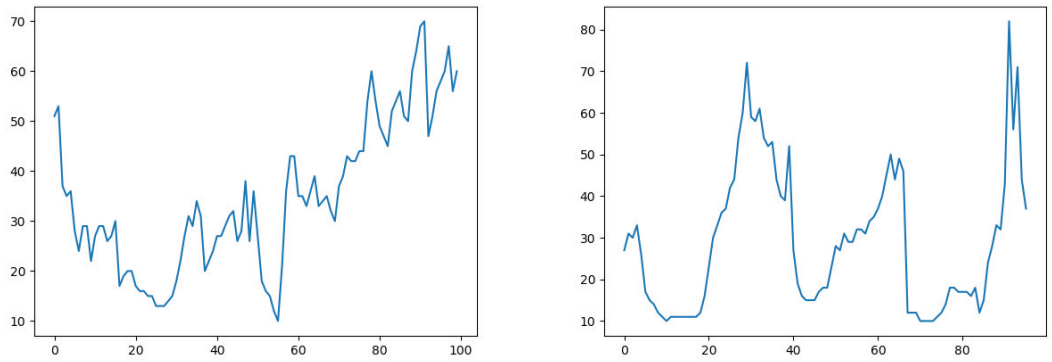
(d) UMD Distances

Fig. 4.12 Distances within the Communities.

4.2.5.3 Number of Clusters

In classical semi-supervised clustering algorithms, the number of clusters is directly extracted using the classes provided as input by the user. This solution is limited since the user who might be interested in finding a different number of clusters from those in the supervised dataset cannot really obtain those through the clustering step. We show that the third parameter, called 'Number of Clusters', allows obtaining this behavior. Therefore, with the third parameter, it will be possible to choose the number of final clusters the user wants to visualize.

The importance of this parameter is twofold. The first meaning is the possibility of finding undefined classes in the supervised dataset, providing additional information about the input dataset and further insights. Moreover, this parameter combines all the communities of global relations into a precise number of clusters. Indeed, the number of communities detected may be different in each feature. Therefore, defining a precise number of clusters favors the relationships obtained from a feature with several communities equal to the number



(a) Acute Dialysis (b) Chronic Dialysis

Fig. 4.13 Time series of the GFR signal for patients treated for Acute and Chronic Dialysis.

of clusters. Thus, the last two steps of the pipeline are dedicated to this parameter, as shown in Figure 4.9.

We show the importance of deciding upon the number of clusters to be extracted despite a different number of labels provided as input. The GFR dataset shows two classes called Kidney Failure and Not Kidney Failure. With the help of a domain expert, we have increased the number of classes to analyze the behavior of FeatTS with classes different from those given as input. After interacting with the expert, we obtained two other classes out of the Kidney Failure class, namely Chronic Dialysis and Acute Dialysis. Patients under chronic dialysis must regularly go to the hospital for health care. Chronic kidney disease is a disease in which there is gradual loss of kidney function over months to years. Acute kidney injury is an abrupt loss of kidney function that develops within seven days. An acute kidney injury occurs when the kidneys are exposed to something harmful or considered harmful by the body.

In this experiment, we show the usefulness of FeatTS to help the medical doctors separate the cases of chronic dialysis from the acute dialysis while not having these labels available as input. Figure 4.13 shows the two classes discovered by FeatTS.

4.3 Time2Feat: Clustering Multivariate Time Series

We introduce Time2Feat, an open-source scalable and interpretable system for MTS clustering that adopts an end-to-end semi-supervised clustering pipeline, mainly based on feature extraction, feature selection, and clustering. Features are automatically extracted from the signals composing the MTS by exploiting both *intra-signal features*, characterizing the

single signals of MTS, and *inter-signal features* measuring pairwise relatedness (in terms of similarity and correlation) of multiple signals employing interpretable metrics. Two dataset-dependent techniques are introduced to select the most important features among the ones describing the MTS. The *unsupervised mode* is an entirely intuitive approach based on Principal Features Analysis (PFA) [82]. The *semi-supervised mode* relies on users' annotations on small dataset samples to improve the selection process. Finally, a clustering technique is adopted for a group of the MTS. The resulting clusters turn to be *interpretable*: users can conduct an in-depth analysis of them to understand why MTS share the same cluster.

4.3.1 Motivating Real-World Scenario

The BasicMotions dataset is a real-world dataset belonging to the UEA multivariate time series classification archive [7]. This dataset describes four kinds of activity (i.e., playing badminton, running, standing, and walking) performed by students through two sensors (an accelerometer and a gyroscope) installed in their smartwatches. The sensors gather data in a three-dimensional space, thus producing three different signals (X, Y, Z). The overall dataset comprises 80 MTS, and each signal includes 100 recordings. We use this dataset to provide an example motivating our research. Suppose we were asked to analyze the dataset without detailing the activities that MTS describes. This lack of information frequently happens in business scenarios where trade secrets or simply the costs for labeling make it necessary to work with unlabeled datasets. Clustering is one of the main data exploration techniques we can perform on unlabeled data.

Generating clusters for MTS is a non-trivial task. From a data structure perspective, they are third-order tensors, i.e., a dataset includes many MTS, each one containing multiple signals, and each signal is composed of several timestamps. From a numerical perspective, it is frequent to work with datasets composed of thousands of MTS and hundreds of signals with thousands of records (see, for example, the datasets used in the experiments in Section 4.3.3). This problem gives rise to a first challenge to address: **(C1):** *Analyzing MTS datasets requires the application of scalable techniques capable of dealing with the high dimensionality of the data.*

We address this challenge by proposing Time2Feat, which computes the clusters based on features extracted from the signals of the MTS. This operation allows us to reduce the problem's dimensionality: from the many timestamps constituting the time series to the single values of the features. We rely on external specialized software libraries to extract *intra and inter-signal* features from the MTS. The former describes particular properties of the signals in isolation (e.g., the mean value, the autocorrelation, et cetera.). The latter evaluates pairwise

the signals measuring distances, correlations, et cetera. This operation overall generates 4842 features in the BasicMotion Dataset. Nevertheless, the high dimensionality of the features extracted could introduce inefficiency in the cluster generation process. A scalable approach should identify the most important features and base the clustering technique on them. In addition, high dimensionality datasets generate ineffective results. Clusters of elements resulting from high-dimensional datasets are difficult to manage and understand for humans. Although very close according to some statistical quality metrics (e.g., Silhouette [34]), the clusters would be unusable by users who would not understand the reasons why the elements were grouped together. This problem introduces a second challenge: **(C2):** *providing an interpretable clustering technique is of paramount importance for data analysis.*

State-of-the-art approaches for MTS clustering [76, 75, 58] suffer from low interpretability. Time2Feat addresses this problem by letting the approach use a reduced number of interpretable features. In particular, Time2Feat adopts a mechanism for the feature selections based on the PFA, that allows us to rank the features according to their importance in the process and to select only the meaningful ones. Table 4.14a shows the features adopted for clustering reduced from PFA to 46 from the 4842 initially extracted.

The number of clusters to generate is another critical parameter to select. We could adopt the well-known Elbow Method to automatically compute the number that better fits the computed statistical measures. The application of the Elbow method to BasicMotion generates 4 clusters, as shown in Figure 4.14c in blue circles. This result is obtained through a completely automatic *unsupervised procedure* where the pipeline starts with the BasicMotion dataset and generates 4 clusters generated via 46 features. Data analytics processes are typically the result of several iterations where users gain more and more insights from the data that require the application of deeper analytical functions. This intuition is the third challenge to address: **(C3):** *clustering techniques for data analytics need to put the human in the loop.*

Time2Feat addresses this challenge by allowing users to select the number of clusters and label some samples for each, thus also supporting a *semi-supervised procedure*. Our experiments demonstrate that just labeling a few elements per cluster improves the accuracy of the results and significantly reduces the number of features adopted by the clustering technique, thus improving their interpretability. Going back to our example, suppose that a user decides to manually inspect the clusters and provides four labels for each cluster. For instance, the user can analyze some elements from the top-right cluster of Figure 4.14c and observe that they refer to people playing badminton. By analyzing some elements from the second top-right cluster, s/he can observe that they refer to people doing running. Time2Feat exploits the user annotations by further reducing the number of features used

for the clustering (from 46 to 3), as shown in Table 4.14b, and improving the quality of the clusters. Figure 4.14d shows the result of the clustering process where a subset of elements in the dataset has been annotated. We observe that the limited number of features used for the clustering allows the user to understand why the data have been grouped (i.e., they are describing the same kind of activity).

Finally, we would like to point out that cluster analysis offered by Time2Feat is exceptionally flexible and facilitate more precise and flexible data exploration. For instance, in case the user wants to group the data into two clusters instead of four, Time2Feat would obtain the first cluster with blue and green data points in Figure 4.14d (the one is referring to the badminton and running activities) and the second with gray and red elements (referring to the standing and walking activities).

4.3.2 Time2Feat Pipeline

Time2Feat implements the components of the data analysis pipeline as illustrated in Figure 4.15. Time2Feat takes MTS dataset D as input and the number of clusters to generate (provided by the user or via some heuristic). It can run under *unsupervised mode*, i.e., no further input is required, and under *semi-supervised mode*, i.e., the users specify a subset of clustered samples. The three steps of the pipeline are implemented by the *feature_extraction* function (described in Section 4.3.2.1), the *feature_selection* function (in Section 4.3.2.2) and the *cluster* function (in Section 4.3.2.3).

4.3.2.1 Feature Extraction

The goal is to generate an exhaustive representation of an MTS dataset via a large spectrum of features, each describing the MTS signals (in isolation or pairs).

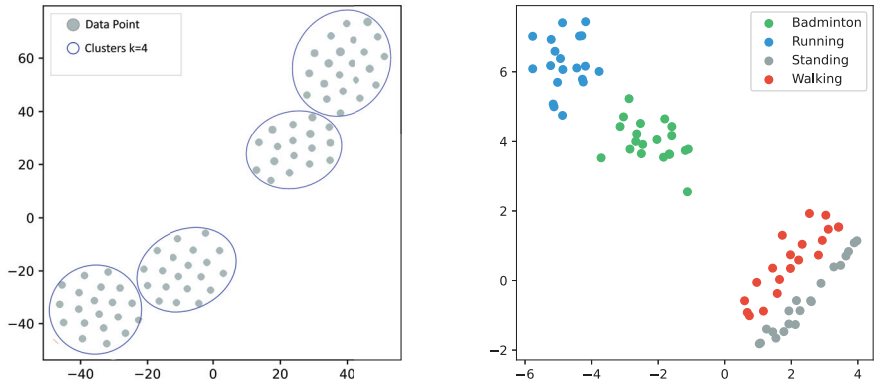
Intra-signal Features Extraction. The computation of statistical features describing the signals of the MTS relies on the library `tsfresh` [30], already adopted in many time series analysis tasks [115, 149, 116]. Out of the 700+ computed by `tsfresh`, each feature encodes the signal description from the perspective offered by a specific analysis method, such as Distribution Analysis, Statistical Analysis, et. Lines 2-6 of Algorithm 1 shows a simplified procedure where a nested for-cycle is used to iterate over the MTS in the datasets and applies the `intra_feature_extraction` function to generate the features for each composing signal. In the actual implementation, we leverage the efficient parallelization of the feature extraction function provided by `tsfresh` that can compute features in batches of univariate series.

	Intra-Signal						*Inter-Signal*		
	AccX		AccY		AccZ	GirX	AccX-AccY		AccX-AccZ
	Quantile0.4	...	PACF 9	...	...	...	L_1 dist.	Chebyshev	L_1 dist.
MTS_R	1.87	...	0.16	...	...	...	981.78	49.56	1115.71
MTS_B	4.45	...	0.26	...	...	...	1121.45	30.71	1423.28
MTS_S	-0.19	...	-0.23	...	...	...	85.05	1.93	35.60
MTS_W	0.71	...	0.14	...	...	...	301.46	8.45	174.99
...	...	...	...	...	...	...	...	...	...

(a) Excerpt of the features (46) extracted in the unsupervised mode.

	Intra-Signal		*Inter-Signal*
	AccX	GirX	AccY-AccZ
	Variance	Quantile 0.3	L_1 dist.
MTS_R	80.12	-0.94	764.99
MTS_B	139.75	-3.57	1083.85
MTS_S	0.09	-0.14	97.05
MTS_W	2.26	-1.02	325.82
...	...	...	...

(b) The features (3) extracted in the semi-supervised mode.



(c) Clusters generated with the unsupervised mode.

(d) Clusters generated with the semi-supervised mode.

Fig. 4.14 The clustering analysis supported by Time2Feat on the BasicMotion dataset.

Inter-signal Features Extraction Many works [78, 124] highlight the importance of inter-signal relationships in the analysis of time series. Nevertheless, they typically extract the features employing neural network architectures, obtaining uninterpretable descriptions. We adopt a straightforward approach by conceiving inter-signal features as the measure of the relatedness (in terms of similarity and correlation) between pairs of signals that we measure through 8 metrics (e.g., correlation, Euclidean distance, et cetera). The pipeline firstly generates the pairs of signals per time series (lines 7-12 of the Algorithm 1), then applies the function `inter_feature_extraction` in charge of the extraction.

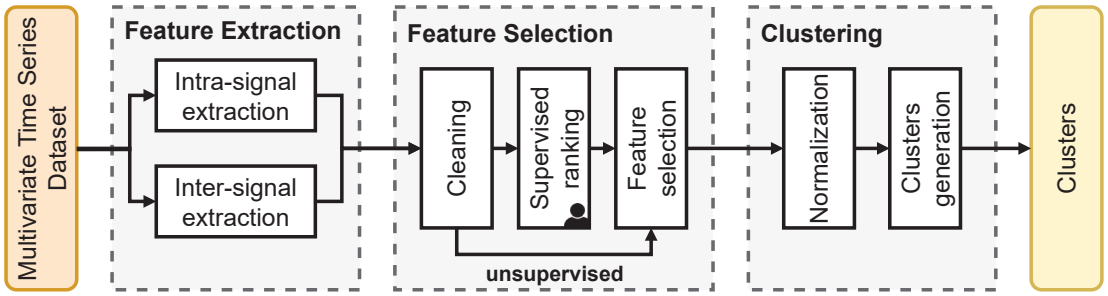


Fig. 4.15 The proposed pipeline and the implementation in the Time2Feat system.

Algorithm 1: `feature_extraction`

Input : $D \in \mathbb{R}^{VxNxS}$ Multivariate time series dataset.
Output : $F \in \mathbb{R}^{VxE}$ Matrix of signals and extracted features.

```
// Extracting intra-signal features
F[] ← 0; // list of extracted features
foreach V ∈ D; // For each MTS in the dataset
do
    foreach S_i ∈ V; // For each signal in the MTS
    do
        F ← intra_feature_extraction(S_i)
// Extracting inter-signal features
foreach V ∈ D do
    foreach S_i ∈ V do
        V = V − S_i;
        foreach S_j ∈ V; // For pairs of signals in the MTS
        do
            F ← inter_feature_extraction(S_i, S_j)
return F;
```

The feature extraction procedure generates a large number of features per dataset. Reducing the dimensionality of such representation improves the interpretability and increases the performance of the clustering procedure.

4.3.2.2 Feature Selection

As a first operation, in line 1 of Algorithm 2, we `clean` the matrix of the features by removing all zero-variance features and the features that have missing or infinite values (they would be useless for the cluster generation). Suppose there are labels available, i.e., Time2Feat is running with the *semi-supervised mode*. These are used to rank the relevance of the features for identifying a subset capable of generating clusters. We rely on the *Analysis of Variance*

Algorithm 2: `feature_selection`

Input : $F \in \mathbb{R}^{VxE}$ Matrix of signals and features.
labels Optional labels
Output : $t \in \mathbb{R}^{VxF}$ Matrix of signals and top features.
`// Remove features with constant, null and/or infinite values`
1 $T \leftarrow clean(F)$;
`// Semi-supervised step if the user labels some records`
2 **if** *labels* **then**
3 $T \leftarrow auto\_anova\_selection(T, labels)$;
4 $T \leftarrow pfa(T)$; `// Extract best feature with PFA`
5 **return** T;

(ANOVA) [128] for computing the p-value associated with each feature and quantifying its significance. Then, we apply a grid search analysis identifying the subset of features that maximizes the quality of the generated clusters. To evaluate the quality, we use the *Homogeneity Score* which measures the homogeneity of the elements in a cluster by using conditional entropy analysis, the Adjusted Mutual Information (AMI) [113] and *Adjusted Rand Index*[129] which are specific measures evaluating the agreement and similarity of pairs of cluster elements. The joint application of the ANOVA and grid search analysis is referred to `auto_anova_selection` in line 3 of the Algorithm. Finally, both in the presence and in the absence of labels, we apply the PFA technique to reduce the dimensionality of the feature representation further. PFA is a variation of the PCA preserving the original values of the features and then the distance between them.

4.3.2.3 Clustering

Concerning the number of clusters, the Time2Feat system leverages state-of-the-art heuristics (e.g., applying the well-known Elbow method) or user preferences. Moreover, in Section 4.3.3, we show the evaluation of classical clustering algorithms, among which the Hierarchical technique achieved the best accuracy in the results. Finally, the clustering operation includes a normalization step that avoids the dominance of features due to large-scale domain ranges.

4.3.3 Experimental evaluation

The evaluation addresses four main research questions:

RQ1 How effective is Time2Feat in solving MTS clustering tasks (Section 4.3.3.1);

RQ2 To what extent the clusters generated are interpretable (Section 4.3.3.2);

RQ3 How efficient is the cluster computation (Section 4.3.3.3);

RQ4 How robust is the pipeline, i.e. to what extent do the components in the pipeline contribute to the task. (Section 4.3.3.4)

Baselines. We selected 18 benchmark datasets from the UEA multivariate time series classification archive [7]. For each dataset, Table 4.5 reports the number of MTS (V), the number of signals (S), the length (N) of the series, and the clusters (C), where the MTS can be grouped. In addition, we computed the overall number of elements in the dataset (E_O – obtained by multiplying $V \times S \times N$) that provides a yardstick for measuring the scalability of the approach. Finally, we estimate the complexity of generating the clusters by computing the number of elements per MTS (E_M – obtained by multiplying $S \times N$). Intuitively, the lower the value, the lower the ability to extract descriptive features. The datasets represent different scenarios as their overall number of elements E_O spans over three orders of magnitudes, and E_M ranges from 16 elements for the PD dataset to 10000 for SW.

We compared Time2Feat with seven approaches: Hierarchical, KMeans, and Spectral are straightforward applications of these classical clustering techniques to MTS datasets. CPSCA and *MC$_2$PCA* introduce a PCA-based mechanism to reduce the data dimensionality before the clustering. DETSEC leverages neural networks by creating embeddings for the series through autoencoders, and DTW measures the similarity between two temporal sequences.
Setup. The experiments are executed on an Intel Xeon Processor machine with 12 cores, 64GB of RAM, and 324GB of local (SSD) storage. The machine runs Ubuntu version 18.04. All experiments have been executed ten times, and the average result plus standard deviation is reported.

4.3.3.1 Effectiveness

To measure the accuracy, we run Time2Feat on the datasets in the benchmark. We computed the AMI, which takes a value of 1 when the evaluated clustering is identical to the baseline, and 0 when the different cluster distributions are random. Table 4.6 shows the results of this experiment. Time2Feat has been evaluated executing the *unsupervised mode* (column **T2F**$_0$) and by providing a stratified random sample composed of 20% (column **T2F**$_2$), 40% (column **T2F**$_4$), 50% (column **T2F**$_5$) of labels per cluster from the baseline datasets. The latter has been used to simulate the user interaction in the *semi-supervised mode*. The remaining columns show the competing approaches (discussed in Section 5.1.1). Among them, Hierarchical, KMeans, and Spectral can be considered as reference baselines for their simplicity[2]. Finally, in Table 4.6, we mark in bold the best value per dataset, and with ↑ the results where the selected Time2Feat configuration overcomes the competing approaches

[2]We rely on the `sklearn` implementations of these algorithms with default parameters.

Dataset	V	S	N	C	E_O $(VxSxN)$	E_M (SxN)
Li — Libras	360	2	45	15	32400	90
AF – AtrialFibrillation	30	2	640	3	38400	1280
BM – BasicMotions	80	6	100	4	48000	600
RS – RacketSports	303	6	30	4	54540	180
ER – ERing	300	4	65	6	78000	260
Ep – Epilepsy	275	3	206	4	169950	618
PD – PenDigits	10992	2	8	10	175872	16
SW – StandWalkJump	27	4	2500	3	270000	10000
UW – UWaveGestureLibrary	440	3	315	8	415800	945
Ha – Handwriting	1000	3	152	26	456000	456
AW – ArticularyWordRecognition	575	9	144	25	745200	1296
HM – HandMovementDirection	234	10	400	4	936000	4000
LS – LSST	4925	6	36	14	1063800	216
Cr – Cricket	180	6	1197	12	1292760	718
EC – EthanolConcentration	524	3	1751	4	2752572	5253
S1 – SelfRegulationSCP1	561	6	896	2	3015936	5376
S2 – SelfRegulationSCP2	380	7	1152	2	3064320	8064
PS – PhonemeSpectra	6668	11	217	39	15916516	2387

Table 4.5 Datasets used in the experiments. V is the number of MTS, S the number of signals, N the length of the series, C the number of classes, E_O the overall number of elements per dataset, E_M the number of elements per MTS.

while not obtaining the best accuracy value. We do not consider the confidence intervals due to the high discrepancy between the computed values.

Discussion. The experiment results clearly show that Time2Feat outperforms its competitors. In particular, in the unsupervised mode, the accuracy of the clusters generated by Time2Feat is higher than the other approaches in 12 out of 18 datasets. Among them, in 3 datasets, it obtains the best accuracy score. By providing 20% labels per cluster, Time2Feat outperforms the other approaches in 13 datasets (obtaining in 2 datasets the best accuracy score). The performances generally improve by adding more labels, as in the configuration $\mathbf{T2F}_5$, where Time2Feat outperforms the other approaches in 15 out of 18 datasets (showing the best accuracy value in 9 out of 18 datasets). This experiment helped us derive the following insights: (1) Time2Feat's pipeline is highly efficient as at least one configuration of Time2Feat outperforms the other approaches in all datasets, except in the UW dataset, where it performs slightly worse than some competing approaches. The reason is that UW describes trajectories. One kind of trajectory is the composition of two other trajectories. The features extracted by Time2Feat cannot recognize these three different movements. (2) Time2Feat is highly scalable as it obtains high accuracy results both for small (the ones at the top of Table 4.6) and for large datasets (the ones at the bottom of the Table 4.6). These results do not hold for the competitors, where the accuracy drops as the number of elements in the dataset increases

Dataset	Semi-supervised			Unsupervised	Competing approaches						
	T2F$_2$	**T2F$_4$**	**T2F$_5$**	**T2F$_0$**	**Hierarchical**	**KMeans**	**Spectral**	**DTW**	**CPSCA**	**DETSEC**	**MC$_2$PCA**
Li	0.728±0.02↑	0.722 ± 0.016↑	**0.730±0.020**	0.716±0.012↑	0.563	0.545	0.492	0.503	0.311	0.416	0.069
AF	0.028±0.046	0.123 ± 0.066↑	**0.238±0.059**	0.038±0.027↑	-0.002	-0.002	-0.002	0.005	-0.07	-0.001	-0.056
BM	0.977±0.034↑	**1.000 ± 0.000**	**1.000±0.000**	**1.000±0.000**	0.347	0.23	0.002	0.832	0.7	**1.000**	0.189
RS	0.559±0.038↑	0.666 ± 0.049↑	**0.710±0.047**	0.35±0.006↑	0.192	0.194	0.0	0.215	0.221	0.224	0.094
ER	0.801±0.016	0.823 ± 0.014	0.826±0.023	**0.921±0.011**	0.859	0.91	0.0	0.775	0.5	0.646	0.115
Ep	0.896±0.025↑	**0.913 ± 0.019**	0.882±0.007↑	0.792±0.04↑	0.135	0.167	-0.001	0.25	0.258	0.213	0.08
PD	0.752±0.022↑	0.771 ± 0.028↑	**0.784±0.013**	0.437±0.02	0.728	0.682	N/A	0.6	N/A	0.431	0.065
SW	0.038±0.036	0.101 ± 0.01	**0.23±0.046**	0.048±0.079	0.131	-0.002	0.0	-0.005	-0.072	-0.097	0.045
UW	0.555±0.026	0.554 ± 0.036	0.59±0.035	0.587±0.055	**0.752**	0.712	0.0	0.611	0.236	0.414	0.111
Ha	0.325±0.023↑	**0.353 ± 0.019**	0.349±0.009↑	0.161±0.006	0.226	0.193	0.0	0.235	0.165	0.271	-0.004
AW	0.921±0.007	0.931 ± 0.01↑	0.927±0.005↑	**0.963±0.007**	0.926	0.902	0.0	0.781	0.716	0.794	0.182
HM	0.021±0.011↑	**0.045 ± 0.007**	0.07±0.012	0.015±0.008	-0.006	-0.002	0.001	0.01	0.002	-0.004	0.018
LS	0.293±0.013↑	0.317 ± 0.011↑	**0.333±0.002**	0.156±0.009↑	0.028	0.018	0.001	N/A	0.047	0.152	0.048
Cr	**0.984±0.021**	0.975 ± 0.018↑	0.974±0.021↑	0.946±0.021↑	0.756	0.719	0.0	N/A	0.876	0.865	0.361
EC	0.065±0.017↑	0.097 ± 0.006↑	**0.121±0.04**	0.052±0.002↑	0.009	0.01	-0.003	N/A	0.013	N/A	0.002
S1	**0.397±0.047**	0.374 ± 0.025↑	0.382±0.012↑	0.007±0.001	0.212	0.194	-0.001	N/A	N/A	0.18	0.022
S2	0.008±0.003↑	**0.015 ± 0.004**	**0.015±0.006**	0.003±0.001 ↑	-0.002	-0.001	0.01	N/A	0.001	0.007	0.005
PS	0.2±0.006↑	**0.202 ± 0.002**	0.201±0.002↑	0.121±0.007↑	0.093	0.096	0.0	N/A	N/A	N/A	0.058

Table 4.6 Effectiveness (AMI score). In bold, the best value per dataset. The ↑ mark denotes Time2Feat settings that overcome the competing approaches.

(and in some cases, marked N/A in the Table 4.6, no cluster is generated due to timeout or memory exceptions). (3) The semi-supervised procedure involving the user annotations improves the accuracy. By labeling a small number of elements per dataset, the accuracy steadily increases.

4.3.3.2 Interpretability

We assume that the user's perception of a few features is better than the user's exploration of a large number of features. Therefore, we measure the interpretability of the clusters as the number of features that Time2Feat uses for their computation. The column All in Table 4.7 shows the overall amount of features extracted after the feature extraction step of the pipeline (Section 4.3.2.1). The other columns report the number of features retained with the unsupervised mode (column **T2F$_0$**) and with increasing levels of supervision as in the previous experiment. The values represent the average of the features selected in the ten experiments.

Discussion. The feature extraction generates a large number of features that increases as the number of elements per dataset E_O (the Pearson correlation coefficient – Pcc is 0.61) and the number of elements per MTS E_M (Pcc = 0.39) increases. The unsupervised approach drastically reduces the selected features while maintaining a strong correlation (Pcc = 0.51) with the overall number of features. Not always an increase in the supervision corresponds to a reduction in the features. We explain this as a sort of "overfitting" that forces better accuracy results by adding features. However, we observe the small number of features retained in all semi-supervised settings that allow users to interpret the cluster generation

Dataset	All	$\mathbf{T2F}_0$	$\mathbf{T2F}_2$	$\mathbf{T2F}_4$	$\mathbf{T2F}_5$
Li	1574/8	55.0/1.0✓	7.17/0.0	8.33/0.0	9.4/0.0
AF	1574/8	21.0/0.0	2.83/0.17	5.67/0.0	5.67/0.0
BM	4722/120	44.33/1.67✓	2.33/1.5✓	2.0/0.67	2.0/0.17
RS	4722/120	141.2/9.8✓	12.8/2.6✓	15.4/3.4✓	21.0/4.2✓
ER	3148/48	125.83/3.17✓	7.0/1.67✓	6.83/1.33✓	7.17/1.17✓
Ep	2361/24	163.33/3.67✓	12.67/1.83✓	15.67/1.17✓	15.33/1.33✓
PD	1574/8	98.0/1.0✓	16.4/0.6	13.8/0.6	18.8/0.8
SW	3148/48	20.0/0.0	1.8/0.4	3.4/0.0	6.4/0.0
UW	2361/24	124.0/3.0✓	4.4/0.2	4.4/0.2	4.4/0.0
Ha	2361/24	309.83/3.17✓	23.83/1.5✓	25.0/2.17✓	30.83/2.67✓
AW	7083/288	283.67/30.33✓	10.0/5.0✓	9.5/4.0✓	10.5/4.0✓
HM	7870/360	167.0/11.0✓	15.17/1.0✓	28.17/1.33✓	24.83/2.17✓
LS	4722/120	217.0/5.0✓	6.6/3.6✓	9.4/3.2✓	12.8/4.4✓
Cr	4722/120	113.17/3.83✓	4.5/3.83✓	4.17/4.17✓	4.17/4.0✓
EC	2361/24	122.83/5.17✓	9.33/0.33	8.5/0.0	3.0/0.0
S1	4722/120	222.2/1.8✓	2.0/0.2	2.0/0.0	3.0/0.2
S2	5509/168	183.0/3.0✓	26.4/0.4	20.8/0.4	20.2/0.0
PS	8657/440	293.0/8.0✓	4.0/0.0	4.4/0.0	4.2/0.0

Table 4.7 Interpretability as number of intra-signal / inter-signal features. ✓ indicates 1+ inter-signal feature(s).

process. Finally, the experiment highlights the importance of the inter-signal features. Even if, by construction, the number of inter-signal features is lower than the intra-signal, the final retained features include inter-signal elements for almost all Time2Feat settings, thus showing their importance in the clustering process.

4.3.3.3 Efficiency

We perform three experiments to evaluate the efficiency of our approach. The first experiment, called *Time Performance*, measures the efficiency by computing the overall time required to complete the pipeline. The second experiment, called *Time breakdown of the pipeline components*, evaluates the time breakdown of Time2Feat's pipeline. Finally, the third experiment, called *Workload balancing*, improves the pipeline by introducing a simple heuristic to optimize the parallelism of the feature extraction.

Time Performance Table 4.8 shows the maximum time to complete the cluster computations for all the datasets in the 10 experiment repetitions. We show only the time measured in the unsupervised mode ($\mathbf{T2F_0}$): the semi-supervision does not change the value significantly. The last row shows the average time computed on all datasets (excluding the ones raising the exceptions).

Discussion. The clustering techniques that reach the best time performance are KMeans and Hierarchical, which finish the pipelines in a few seconds. However, this comes at the cost

Dataset	**T2F**$_0$	Hierarch.	KMeans	Spectral	DTW	CPSCA	MC_2PCA	DETSEC
Li	20.31	0.2	0.28	0.37	366	2	0.53	500
AF	31.01	0.04	0.06	0.31	350	0.01	0.12	876
BM	58.45	0.09	0.16	0.23	175	0.03	209	260
RS	50.11	0.2	0.39	0.39	474	0.317	356	270
ER	34.2	0.18	0.24	5.27	914	0.21	559	555
Ep	47.95	0.24	0.42	7.71	3667	0.31	682	1673
PD	198.03	9.0	3.0	×	24713	50	6	3395
SW	559.69	0.16	0.25	0.34	10768	0.01	1	10142
UW	58.42	0.45	0.74	15.6	20639	0.47	3063	4229
Ha	44.74	0.86	2.0	7.19	27018	0.19	25	4301
AW	135.18	1.0	1.0	1.35	23611	1	18811	220
HM	220.78	0.83	1.0	0.89	30251	0.62	3162	3175
LS	300.23	6.0	3.0	6.49	-	0.35	16591	4666
Cr	737.61	0.8	1.0	0.91	-	0.14	13642	12145
EC	876.3	2.0	2.0	2.01	-	0.26	9708	-
S1	727.37	2.0	2.0	2.37	-	-	12	19317
S2	952.58	2.0	2.0	2.22	-	0.36	11	20898
PS	1219.88	1.0	1.0	34.73	-	×	×	-
Avg	348.49	1.50	1.14	5.19	11912.17	3.52	3931.69	5537.88

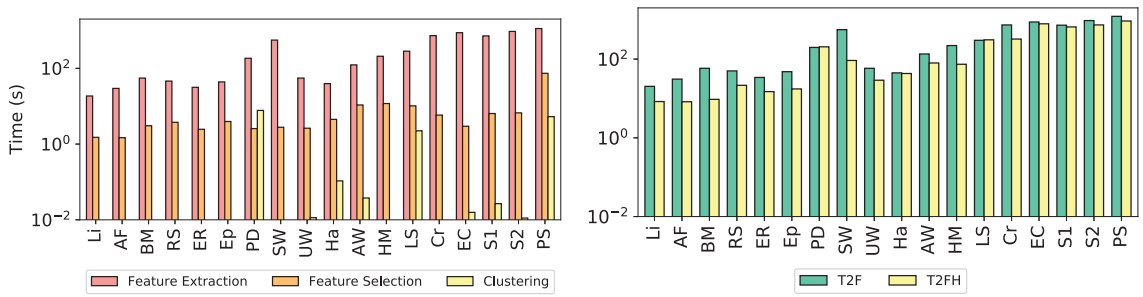
Table 4.8 Runtime execution , in seconds (- timeout exception fixed in 10 hours, × memory exception).

of accuracy and interpretability loss. CPSCA also shows a low time performance, but the algorithm cannot handle large datasets, where time and memory exceptions occur and poor accuracy. The other approaches report an average time greater than Time2Feat of at least an order of magnitude and, in some cases, time and memory exceptions. Finally, we observe that Time2Feat's execution time ranges from a few seconds to a few thousands of seconds, having the performance correlated with the overall number of elements (Pcc=0.75).

Time breakdown of the pipeline components The goal of this experiment is to analyze the breakdown of the computation time (see Figure 4.16a) into the main pipeline components (feature extraction, feature selection, and cluster generation).

Discussion. In all datasets, the time required for extracting the features dominates the other components: it takes between 88% and 99% of the overall time needed for completing the pipeline. The average time to complete the feature extraction is around 337 seconds, and the one to complete features selection is 9 seconds, whereas, for clustering, it amounts to 1 second. The correlation between the time spent in clustering and V is strong (Pcc=0.99). This is why in three datasets (PD, LS, and PS, the ones with the largest V), the clustering time takes more than 1 second but less than 8 seconds. The feature extraction and selection show a different behavior: they are correlated with the overall number of elements in the datasets E_O (the Pcc is more than 0.95 for both the tasks).

Workload balancing In this experiment, we evaluate a straightforward heuristic to improve the time performance by optimizing the computational workload on the processors. We recall



(a) Pipeline Breakdown.

(b) Custom parallelization.

Fig. 4.16 Efficiency analysis.

that feature extraction, based on an external library, performs the computation using batches of time series. These batches are not balanced by default. Time2Feat allows to balance the workload by customizing the number of batches per dataset by dividing the total number of MTS (V) by the number of available processors, rounding for excess to the upper integer. Figure 4.16b shows the time reduction by adopting this heuristic.

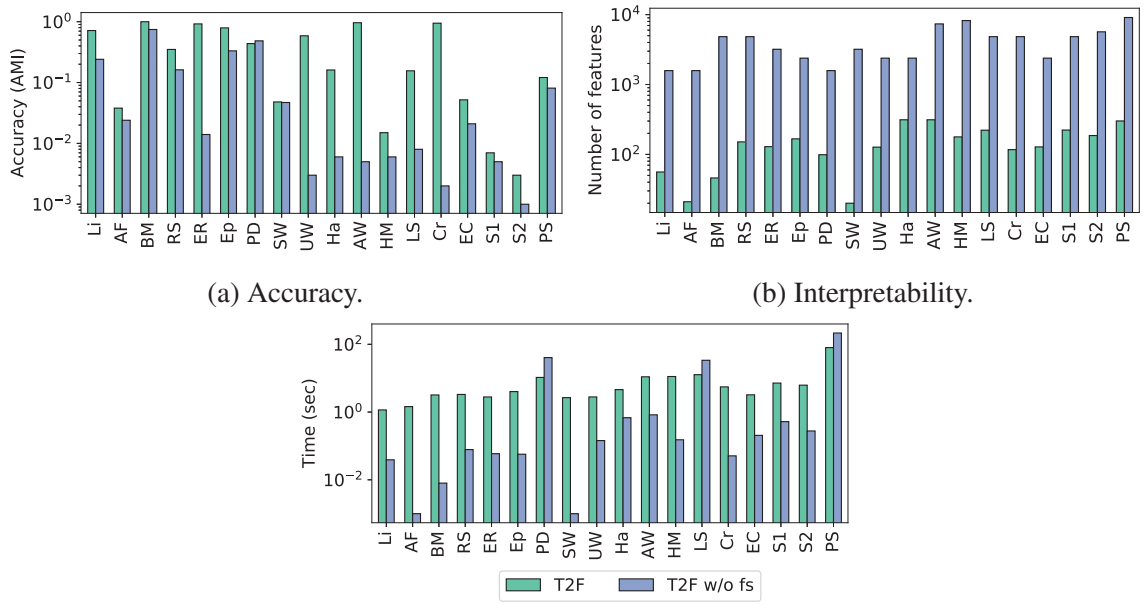
Discussion. By balancing the workload on the processors, the time performance essentially improves in almost all datasets (the average time computed on all datasets decreases from 348 to 242 seconds. In 5 datasets (AF, BM, Ep, SW, and HM), the time reduction is more than 60%. In only two cases (PD and LS), the heuristic does not affect the time performance, and the time slightly increases.

4.3.3.4 Robustness

This Section evaluates the robustness of the pipeline components by a series of ablation tests. We evaluate the importance of feature selection. Then, we evaluate alternative options to the Hierarchical algorithm for performing the final cluster computations. Finally, we evaluate the importance of the features in the clustering procedure.

Importance of feature selection We show the importance of feature selection by evaluating the clusters generated without this operation in terms of accuracy (AMI) in Figure 4.17a, interpretability (number of features) in Figure 4.17b, and efficiency (time for performing the feature extraction and clustering) in Figure 4.17c.

Discussion. The experiment shows that the step removal generally has a large impact on the accuracy and the interpretability. With fewer features, the decrease is two orders of magnitude, Time2Feat obtains more accurate clusters than adopts all the features. The AMI is close to 0 in almost all datasets when the clusters are computed with all features. Concerning the time, feature selection and clustering operation generally take less time than



(a) Accuracy.

(b) Interpretability.

(c) Efficiency for feature extraction and clustering.

Fig. 4.17 Removing the Features Selection from the pipeline.

clustering on all features. The comparison between the time required for executing this step and the duration of the overall pipeline in Figure 4.8 shows that: (1) this time saving is irrelevant, (2) most of the time is spent by the Time2Feat pipeline in the feature extraction step. In summarizing, the feature selection step improves the accuracy and the interpretability, by keeping unaltered the cluster computation time.

Importance of the clustering step We experimented with three techniques (*Hierarchical*, *KMeans*, *Spectral*) for generating the clusters, as shown in Table 4.9. For each technique, we computed the AMI of the clusters obtained with three settings: the *unsupervised procedure* (**T2F**$_0$), the *semi-supervised procedure* with 20% and 50% labeled elements per cluster (**T2F**$_2$ and **T2F**$_5$, respectively).

Discussion. The results show that Time2Feat's default clustering technique (i.e., the Hierarchical) provides the best performance above all the others. The KMeans technique has similar accuracy. The Spectral clustering does not obtain competitive values apart from the most extensive datasets where it achieves the best results, very close to the other approaches.

Importance of the features in the clustering task This experiment evaluates whether a feature-based clustering approach is more effective than an approach based on raw data. To this end, we run Time2Feat in the unsupervised mode by performing the clustering computation with the same techniques used in the previous experiment (Hierarchical, KMeans, and

Dataset	Hierarchical			KMeans			Spectral		
	T2F$_0$	**T2F$_2$**	**T2F$_5$**	**T2F$_0$**	**T2F$_2$**	**T2F$_5$**	**T2F$_0$**	**T2F$_2$**	**T2F$_5$**
Li	0.716↑	0.728↑	**0.730**	0.711	0.691	0.722	0.627	0.709	0.705
AF	0.038↑	0.028	**0.238**	0.007	0.047↑	0.192	-0.001	0.047↑	0.188
BM	**1.000**	0.977↑	**1.000**	**1.000**	0.961	**1.000**	0.992	0.902	0.993
RS	0.35	0.559	**0.710**	0.359	0.578↑	0.649	0.371↑	0.612	0.663
ER	0.921	0.801	0.826↑	**0.955**	0.819↑	0.824	0.925	0.79	0.804
Ep	0.792	**0.896**	0.882↑	0.874↑	0.839	0.867	0.528	0.800	0.793
PD	0.437	0.752↑	**0.784**	0.639↑	0.727	0.715	0.377	0.651	0.688
SW	0.048	0.038	0.231	0.071↑	0.074	**0.327**	-0.004	0.167↑	0.256
UW	0.587↑	0.555↑	**0.59**	0.566	0.541	0.539	0.454	0.511	0.537
HM	0.161↑	0.325↑	**0.349**	0.153	0.302	0.325	0.014	0.277	0.289
AW	**0.963**	0.921↑	0.927↑	0.945	0.918	0.903	0.807	0.89	0.899
HM	0.015↑	0.021	0.069	0.011	0.048↑	**0.089**	0.005	0.037	0.062
LS	0.156↑	0.293	**0.333**	0.146	0.315↑	0.332	0.041	0.037	0.051
Cr	0.946↑	**0.984**	0.974↑	0.907	0.956	0.96	0.787	0.95	0.967
Ec	0.052	0.065	**0.121**	0.056↑	0.066↑	0.094	0.049	0.06	0.106
S1	0.007	0.397	0.382	0.019↑	**0.419**	0.391↑	0.002	0.387	0.379
S2	0.003↑	0.008	0.015	0.000	0.015	0.024	0.000	**0.021**	**0.035**
PS	0.121	0.2	0.201	0.143↑	**0.211**	0.208↑	0.143↑	**0.211**	0.208↑

Table 4.9 Accuracy (AMI) varying the clustering techniques. In bold, the best result per dataset. The ↑ mark denotes the best results per setting (**T2F$_0$**, **T2F$_2$**, **T2F$_5$**).

Spectral), and we compare the accuracy obtained (in terms of AMI) with the one obtained by the application of the same clustering technique to the raw datasets.

Discussion. The comparison with the clustering techniques executed on the original time series shows the feature extraction's importance in obtaining accurate clusters independently from the selected clustering algorithm. Figure 4.18 shows the results obtained in terms of AMI difference between the feature-based and the raw-data-based approaches clustered with the same technique. Positive values mean that the feature-based approach performs better than the raw-based approach, which applies to most datasets. Only three datasets (PD, UW, and S1) show a reduced performance.

4.3.4 Lessons Learned

We conclude by pinpointing how our feature-based clustering pipeline addresses the research aforementioned questions.

(RQ1) Thanks to the features, we gain on effectiveness. The experiment in Section 4.3.3.1 demonstrates that Time2Feat provides more accurate clusters than its competitors. The ablation test confirms that the application of a feature-based clustering technique generates more effective results than the application of clustering on raw data.

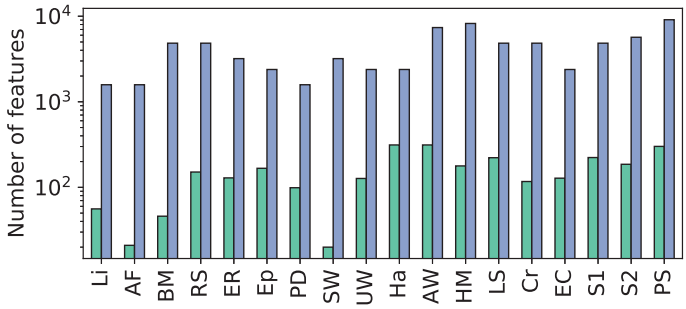


Fig. 4.18 Difference (AMI) between feature-based and raw data clustering with the same technique.

(RQ2) The features facilitate interpretability. The experiment in Section 4.3.3.2 shows that Time2Feat allows exploiting a small number of features for generating the clusters and making them interpretable by the users. Moreover, it demonstrates the importance of inter-signal features retained in many settings for cluster generation.
(RQ3) Feature-based clustering achieves a trade-off between accuracy and performance. Time2Feat achieves the best accuracy in the majority of the datasets along with good performance. The approach is among the fastest ones and performs in seconds, thus making it efficiently usable for batch analyses. The studied time breakdown of the pipeline components shows that the feature extraction phase is the most expensive. Nevertheless, heuristics for optimizing the workload balance, e.g., concerning the available processors, can be quickly developed to reduce the overall time execution considerably.
(RQ4) The pipeline for feature based clustering MTS is robust and scalable. The pipeline is highly modular and then scalable concerning the specificities of real-world environments. Our robustness analysis shows the importance of all components of the pipeline, i.e., shows that feature selection improves both accuracy and interpretability. The use of the hierarchical clustering technique as the default technique, as well as other classical clustering algorithms, shows the adaptativeness of the pipeline, allowing for striking a balance between accurate results in small and large datasets depending on the use case at hand.

4.4 Conclusion

In this chapter, we studied Univariate and Multivariate time series clustering based on interpretable features. First, an interpretable pipeline is proposed for clustering univariate time series based on explainable features and the discovery of global relationships between the series through the usage of the networks. Then, another interpretable pipeline is proposed for clustering multivariate time series. This pipeline is based on two types of interpretable

features (intra-signal and inter-signal) and permits the best results among the multivariate time series clustering algorithms. Both the methods are based on a semi-supervised procedure for selecting the features that improve the latter's performance.

In future work, we aim to improve the multivariate time series clustering to accept the different lengths of signals in time series and implement the global relationship techniques on the signals. In univariate time series, we aim to improve the time performance of the algorithm. Another improvement would be to dynamically choose the threshold for graph creation based on the processed features. Finally, the community detection algorithm's weights could be combined with the features' relevance degrees.

Chapter 5

Time Series Feature-based Anomaly Detection

5.1 Introduction

Anomaly Detection in Time Series data is one of the fascinating recent problems. In particular, with the expansion of the IoT systems, most of the Anomaly Detection problem moves the detection of the inconsistencies on the Real-Time Time Series, i.e., the reception of the points moved its target to the detection online.

Many algorithms solve this problem by considering the raw time series, i.e., without any preprocessing operation on the original data. However, previous studies show interesting approaches for time series data by adapting some characteristics [131, 132]. These articles show an improvement in the algorithm's performance compared with the Raw-Based algorithms. Nevertheless, the data adopted by these approaches is mainly on Offline Time Series, but adapting this preprocessing operation among Online Time Series data can improve algorithms' performance.

It is common in the Time Series online algorithms to find solutions adopting Windowing approaches. This solution permits the observation of only a subset of the data received and avoids recomputing the entire time series on the algorithm. The sliding windows algorithms' performance strongly depends on the window size. Indeed, the window's length is an open problem, and the dimensions strongly depend on the case studies.

AnomalyFeats(AF) is an Online Feature-Based Anomaly Detection algorithm for Time Series data. AF aims to adapt the features to improve the detection of anomalies in Real-Time Time Series using an approach of multi-windows, i.e., creating multiple sizes of windows to

reduce the dependency on the use case studies. AnomalyFeats holistically combines different theoretical approaches to improve each approach's performance, reducing their defeats.

By definition, an anomaly is a measurable result of an unexpected change in the state of a system that exceeds its local or global specification [31]. As already discussed in Section 2.3.4, the most frequently encountered anomalies may be summarized in:

- **Point Anomalies**: The values are entirely too far off from the rest.
- **Contextual anomalies**: Sequences or single instances deviate from the expected patterns of the time series; however, if taken in isolation, they can be within the range of expected values for the time series.
- **Pattern Anomalies**: A collection of anomalous observations concerning the rest of the data.

In addition to detecting these anomalies, an anomaly detection algorithm on streaming data has to be capable of respecting some intrinsic properties of real-time data. Indeed, when the observations arrive constantly, there are some properties that each streaming algorithm has to handle[Thakkar et al.]:

- **Transient**: In the data stream, the importance of a point is directly proportional to the time elapsed since it was detected. It means that an outlier detection algorithm should detect outliers of the observation immediately as it arrives.
- **Infinite**: The reception of the points can be endless; therefore, it appears crucial to handle the memory complexity.
- **Arrival Rate**: The arrival rate can be fixed or variable. The outlier detection technique for data stream has to process data points before the next data point arrives.
- **Concept Drift**: The distribution of the observation can change during the entire reception of the data. Therefore, the algorithm of outlier detection has to adapt to this drift.

AnomalyFeat respects all these properties, adopting a Windowing approach for solving the Transient and Infinite properties. Moreover, AF appears very fast in extracting and evaluating the features. Finally, AnomalyFeats handles the Concept Drift using the Clustering approach and then the Summarization phase.

An important novelty of AF is the ability to holistically combine the most used theoretical approaches in one single methodology.

- The **Forecasting** methods predict the new observation in the nearest future. The prediction of this point aims to discover the expected area of the following original point.
- The **Clustering** helps to detect the shapes of the points distributions. Indeed, as shown in Figure 5.1b, the Clustering can easily understand the position of the point in the distribution compared with the previous points observed.

The comparison of the data needs another technique based on the distances approach. With this technique, AF computes the distances between the evaluating point and his closest point to obtain a final decision about the quality of the last point received.

Another novelty proposed by AF for detecting the anomalies in the Time Series Streaming Data is adopting the features. As shown in [131, 132] and in the previous chapters, the features are a good tool for clustering the time series. Moreover, these characteristics could help make successful studies on the features that best detect an anomaly.

Extracting the features on the time series requires a minimum number of data points. Indeed, if some features are extractable with one single data point, the quality depends on the time series length. In data streaming, the reception of the point is constant; therefore, the time series length increases as more points are received, but AF cannot keep all the points in memory. Moreover, the time required to extract the features tends to increase at each point's reception, violating the arrival rate property.

AF solves the streaming problem by adopting a Novel Multi-Windowing approach. Indeed, AF leverages the windows to extract the features each time new observations come, without saving all the observations arrived. Moreover, AF creates different sizes of sliding windows to reduce the probability of working with unnecessary information due to the wrong sizes of the windows.

5.1.1 Related Work

There are many approaches in the literature for solving anomaly detection algorithms on Online Time Series data. The most straightforward approach adopts some statistical operators as average and variance. The idea is to compute the mean and variance for all the previous data points and fix a tolerance threshold for the anomaly. If an observation exceeds the threshold, the algorithm marks it as an anomaly.

This technique is computationally efficient in terms of time and memory requirements; however, these approaches do not work for most online time series as they mostly ignore the temporal aspects of the data. Moreover, they cannot detect a majority of contextual and collective anomalies[84].

Another similar approach, called Distance Approach, computes the difference between the value of the last observations with the new one. Like the previous approach, if the difference exceeds the tolerance, the Outlier Detection Algorithm notifies the presence of an anomaly. This method also seems to be relatively straightforward. Indeed, it can perform well only in stationary time series data[26].

Anomaly detection algorithms leverage the Clustering to project the observation in a multidimensional space and then compute the density of the observation in the space. The computation of the Density Area around a point helps detect point anomalies. However, Clustering is not applicable in the presence of contextual anomalies. Indeed, if an instance is locally anomalous, i.e., different from the value expected, but globally the instance is known, the Clustering does not detect it as an anomaly[106, 64].

The approach most used by real-time anomaly detection algorithms is undoubtedly the Prediction. With this approach, the time series teaches a regression model[39] to forecast the distribution of the next point. In this way, the algorithm compares the subsequent observations with the forecasted distribution and, in case the deviation is high, it will mark them as anomalies.

This approach is probably the most complete compared with the previous. Indeed, it permits handling all the kinds of anomalies discussed before. The main problems of this approach depend on the number of observations that the model needs to learn the distribution. Teaching regressive statistical models, like ARIMA[139] or SARMA[62], or even worse, the Regressive Neural Network, like GRU[44] or LSTM[43], to recognize perfectly the following points requires a high demand of observations. Therefore, adopting this approach to streaming data requires much time to be accurate.

Finally, the Autoencoder approach now appears as a trending topic. With the Autoencoder, the input data passes into an Encoder algorithm, which tries to summarize the input data as much as possible, creating an object called Code. This Code is the input of another algorithm of the Autoencoder, called Decoder. The Decoder aims to recreate the initial data input based on the object Code, minimizing the difference between original and recreated input[40]. Encoder and Decoder are frequently Neural networks. Indeed, the Encoder is a multilevel neural network, where the number of neurons per level decreases as the levels increase to reduce the number of features required. The final layer outcome represents the Code. The Decoder is even a neural network with the difference that the number of neurons per level increases as the levels increase to reconstruct the initial input [28].

This operation permits recognizing an anomaly when the output of the Decoder is mainly different from the Encoder's input[69]. The Encoder creates a different Code from the ones created during the training. Thus, the Decoder tries to decode a never-seen Code. Therefore,

the expected output of the Decoder will contain many differences from the original input. Hence, this mistake to reproduce the initial input depends on an anomaly of the data in the input.

The results of this approach are exciting, but most of them require much data to train perfectly the Encoder and the Decoder. Moreover, this method appears like a black box, making a deep analysis of the anomalies impossible.

Another strategy adopted in the last years is to use an Ensemble approach. The idea is to use a variety of algorithms to control each observation and apply some form of voting mechanism [107] for the output of each algorithm. An ensemble setup can be composed of similar algorithms, such as a set of Clustering approaches, or different combinations of anomaly detectors, such as Statistical and Predictive Approaches.

This method can improve the performance of the other solutions, but it increases the configuration complexity and the computational time. This increase derives from the manner the algorithms are combined. Indeed, in literature, most ensemble approaches execute the outlier detectors in parallel, without any influence. Subsequently, the voting system has to study each evaluation and then decide. Moreover, this operation requires heavy computation approaches like Predictive and Clustering, a very high time complexity for good performance.

5.2 Anomaly Scoring

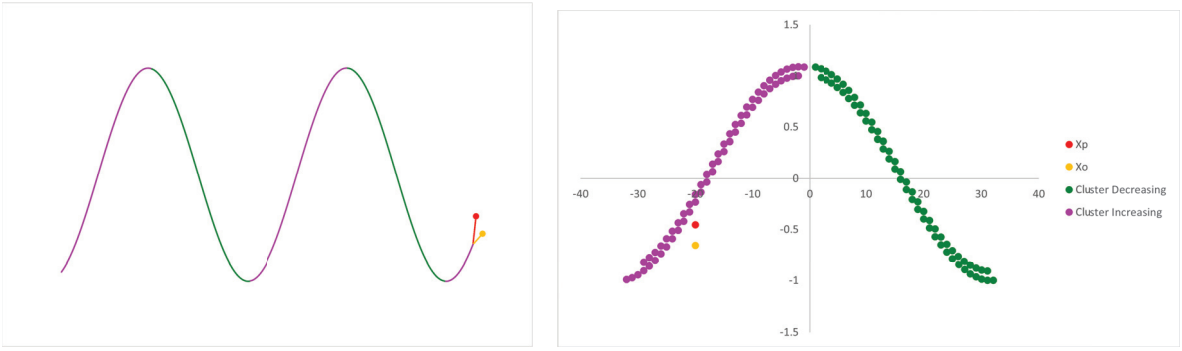
When AF receives a new data point x_o in the time series, the algorithm activates the procedure for estimating the anomaly degree. AF uses a Spatio-Temporal Analysis for evaluating this new point, and the outcome will be named Anomaly Score. The description of the Spatio-Temporal Analysis is subdivided into two principal evaluations:

- In the first evaluation called Context Evaluation, AF estimates the anomaly degree of the x_o adopting the Soft Clustering Algorithm and the predicted point x_p forecasted by the Neural Network.

- In the second evaluation, called Density Evaluation, AF measures the anomaly rate by checking the density area of x_o.

In the Context Evaluation, AF uses the forecasted value x_p for analyzing the new data point x_o. A large number of algorithms[53, 96, 103] gauge the difference between the two points to estimate the anomaly.

AF combines the Predictive or Temporal Analysis with Spatial Analysis to achieve a more effective evaluation. In detail, the idea is to use the forecasted point x_p for discovering



(a) Time Series Received (b) Time Series Clustered

Fig. 5.1 Time Series Example

where x_o has to be placed in the space and then retrieve his cluster, denoted as C_{x_o}, as shown in Figure 5.1b.

In this manner, the information retrieved about all the related data points of the C_{x_o} permits a more in-depth evaluation.

The AnomalyScore formula can be written as follows:

$$AnomalyScore(x_o) = \frac{IntraClusterEvaluation(x_o, x_p, accRate) + InterClusterEvaluation(x_o, x_p) + densityEvaluation(x_o)}{3} \quad (5.1)$$

In AF, as shown in Formula 5.1, there are two different evaluations called *InterClusterEvaluation* and *IntraClusterEvaluation*. These two evaluations are exclusive between them.

The *IntraClusterEvaluation* represents the case when the two points are placed in the same cluster, i.e when C_{x_o} is equal to C_{x_p}. In this situation, the probability of the original point being an anomaly is lower. Thus, AF assigns the 0% of an anomaly for the *InterClusterEvaluation*.

As the opposite, in case the two points are in different clusters, i.e., C_{x_o} is different to C_{x_p}, AF assigns 100% to the *IntraClusterEvaluation*, this is because the probability to be an anomaly is higher.

Analyzing a point's density permits deriving its proximity to the points in the dataset. By definition, a point is in a very dense area when several points with a minimal distance are nearby. The evaluation of the density of a point is essential for detecting anomalies. Indeed, if a point is in a highly populated area, the distribution of values is aligned with it and therefore cannot be considered anomalous.

The following pseudocode summarizes the entire procedure:

Algorithm 3: AnomalyDetectionAlg

Input: Original Point x_o, Clustering Algorithm *clustAlg*, Online Recurrent Neural Network *RNN*, Anomaly Marginal Factor ε

Output: Percentage of Anomaly *AnomalyScore*

Procedure `AnomalyScore(`x_o`)`**:**

- $x_p = RNN.forecast()$
- $C_{x_o} = clustAlg.find(x_o)$
- $C_{x_p} = clustAlg.find(x_p)$
- **if** $C_{x_p} == C_{x_o}$ **then**
  - $varIntraClusterEvaluation = distancePredicted(x_o, x_p) * distanceCentroid(x_o) * membershipStrength(x_o, accRate)$
  - $varInterClusterEvaluation = 1 - \varepsilon$
- **else**
  - $vaIntraClusterEvaluation = \varepsilon$
  - $varInterClusterEvaluation = distanceInterCentroid(x_o)$
- $AnomalyScore = \frac{varIntraClusterEvaluation + varInterClusterEvaluation + DensityEvaluation}{3}$

5.2.1 IntraClusterEvaluation

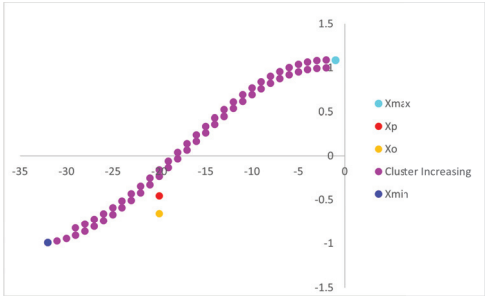
This estimation permits to rate the divergence of the x_o between the data points of the C_{x_o}. The *IntraClusterEvaluation* is based on three main functions:

$$IntraClusterEvaluation(x_o, x_p, accRate) = distancePredicted(x_o, x_p) * distanceCentroid(x_o) * membershipStrength(x_o, accRate) \tag{5.2}$$

$distancePredicted(x_o, x_p)$: This function computes the distance between x_o and x_p. AF evaluates the gap between the x_o and x_p with the gap between the farthest points of the cluster. Indeed, if the difference between the two gaps is very high, it means that the x_o is pretty close to the expected point x_p. Therefore, the Figure 5.2a shows the formula:

$$distancePredicted(x_o, x_p) = \frac{dist(x_o, x_p)}{dist(x_{min}, x_{max})}$$

(a) Equation.



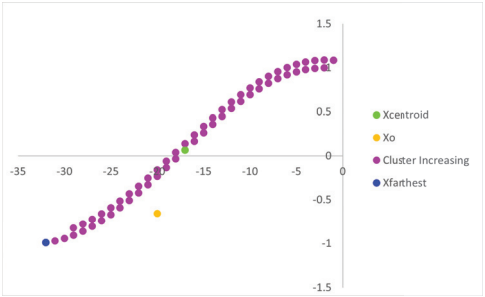
(b) Time Series Clustered

Fig. 5.2 Distance Predicted Formula.

$distanceCentroid(x_o)$: This function evaluates the distance between the original point and the centroid of the C_{x_o} against the maximum distance between the farthest point and the centroid within the cluster. Again, the value of the anomaly grows as the original value moves away from the centroid. Thus, the Figure 5.3a shows the formula:

$$distanceCentroid(x_o) = \frac{dist(x_o,\ x_{centroid})}{dist(x_{farthest},\ x_{centroid})}$$

(a) Equation.



(b) Time Series Clustered

Fig. 5.3 Distance Centroid Formula.

$membershipStrength(x_o)$: With this function, AF allows the user to specify the sensitivity to the anomaly of new points arriving in the time series through a minimum value of membership degree requested to the user.

In particular, each data point received is *Soft Clusterized*, which means that each data point obtains the membership grade of the predicted cluster. These membership grades indicate the degree to which the data points belong to each cluster. Hence, points on the border of a cluster with lower degrees of membership may be in the cluster to a lesser extent than points in the cluster's core.

Therefore, if the new data point has a membership rate below that specified by the user, it is considered an anomaly. Instead, if it is greater or equal than the minimum value indicated, the more it deviates from the minimum value indicated, the less it is an anomaly. Hence, Formula 5.3 is applied:

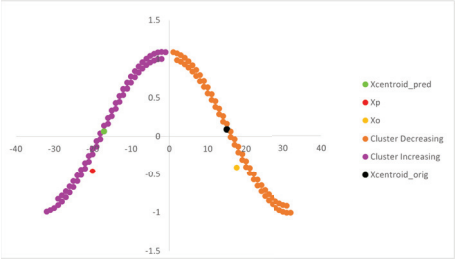
$$membershipStrength(x_o, accRate) = \begin{cases} 1 & \text{if } 0 \leq memberDegree(x_o) \leq accRate \\ 1 - \frac{memberDegree(x_o) - accRate}{1 - accRate} & \text{if } accRate < memberDegree(x_o) \leq 1 \end{cases} \tag{5.3}$$

5.2.2 InterClusterEvaluation

If the predicted and the original point are not within the same cluster, the algorithm analyzes how far the clusters in which the values were settled are from each other. Specifically, AF divides the distance between the centroids of the points clusters by the distance of the centroids belonging to the two most distant clusters.

$$distanceInterCentroid(x_o) = \frac{dist(x_{centroid_o}, x_{centroid_p})}{dist(x_{centroid_1}, x_{centroid_n})}$$

(a) Equation.



(b) Time Series Clustered

Fig. 5.4 Distance Inter Centroid Formula.

5.2.3 Density Evaluation

The purpose of this evaluation is to estimate if the data point x_o is in a highly dense zone. Many algorithms address this problem. AF uses the Local Reachability Density(LRD). LRD is the core of the Local Outlier Factor[21], one of the best algorithms for the detection of anomalies.

The local reachability density is a measurement of the compactness of *k-nearest points* around a point. This value k denotes the number of points close to the point x_o, indicated as $N_k(x_o)$, to analyse. Deciding the value of the k value is not trivial. While a small k has a more local focus, i.e., looks only at nearby points, it is more erroneous when having much noise in the data. However, a large k can miss local outliers. Usually, the choice of the value k depends on the accuracy requested by the user. The value k is also important for computing the *k-distance* that represents the distance between the point, and it's k^{th} nearest neighbour. This distance permits the computation of the Reachability Distance defined as the maximum of the distance between two points and the *k-distance* of that point.

$$reach\text{-}dist(X,Y) = max\ (k\text{-}distance(Y),\ distance(X,Y)) \tag{5.4}$$

If point X is within the k neighbors of point Y, the reach-dist(X, Y) will be the *k-distance* of Y. Otherwise, the distance used will be the actual distance between X and Y.

Once computed all the reachability distances of the *k-nearest neighbors* of the x_o, it is possible to determine his Local Reachability Density (LRD). The local reachability density is a measure of the density of *k-nearest* points around a specific point. LRD employs the inverse of the sum of all of the reachability distances of all the k-nearest neighboring points.

$$LRD(x_o) = \frac{1}{\left(\frac{\sum_{y \in N_k(x_o)} reach\text{-}dist_k(x_o, y)}{|N_k(x_o)|}\right)} \tag{5.5}$$

A high value of $LRD(x_o)$ implies that points are close from the point x_o.

The difficulty in using this technique comes from the impossibility of declaring an interval where the point x_o is close enough to the other points. This problem stems from the fact that LRD can take values ranging from 0 to infinity.

Therefore, AF compares the LRD obtained from x_o with the LRD values obtained from values belonging to its cluster. If the LRD turns out to be in line with the distribution values, then the latter is a non-outlier value. This is because if the value of $LRD(x_o)$ is equal to the distribution of the cluster C_{x_o}, it means that the distance of point x_o with its $N_k(x_o)$ nearest points is equal to the distance of each point x belonging to the cluster C_{x_o} with their k nearest points $N_k(x)$.

To analyze the divergence between the $LRD(x_o)$ from previous LRDs, AF uses a formula known as the Robust Z-Score.

The common Z-score is a measurement that defines a value's relationship to the mean of a set of values. Z-score is measured in terms of standard deviations from the average. If the value of the Z-score is 0, it indicates that the data point's score is identical to the average. A Z-score of 1 would indicate a value of one standard deviation from the average.

The robust Z-Score Method is similar to the Z-score method with some changes in parameters. Since Outliers heavily influence average and Standard Deviation, AF uses Median and Median Absolute Deviation (MAD).

$$Z-score = \frac{x-\mu}{\sigma} \qquad \text{(a) z-score.}$$

$$Robust\ Z-score = \frac{0.6745*(x-Median)}{MAD} \qquad \text{(b) Robust z-score.}$$

Fig. 5.5 Different version of Z-Score.

If x follows a standard normal distribution, the MAD will converge to the median of the half-normal distribution, which is the 75% percentile of a normal distribution. Therefore, to convert this normal distribution into a standard, AF multiplies $N(0.75)$, approximately equal to 0.6745, to the formula.

The choice of the Robust Z-Score derives from the high values that the LRD can assume. Indeed, when two points are extremely close, the LRD computes a value very high. Therefore, using a standard z-score, the average can be completely different from the natural distribution of the LRD. Hence, adopting the Robust Z-Score, which uses the median and the MAD, permits controlling the outliers.

The Robust Z-Score also has another essential property. Indeed, if the values of x follow a standard normal distribution, the Robust Z-Score has the same properties as the distribution. Therefore, it permits the computation of the area under the curve straightforwardly.

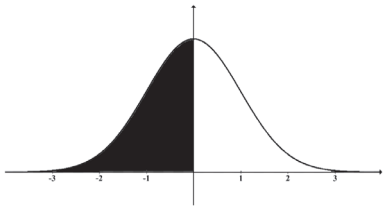


Fig. 5.6 Covered area with Z-Score equal to 0

In fact, in the standard normal distribution, the average is equal to 0 and the variance equal to 1. Thus, the Z-Score (or Robust) can compute the area below the curve in the calculated point. Therefore, as shown in Figure 5.6, in case the value computed by the Robust Z-Score is 0, the area below the curve covers 50% of the total. The Standard Normal Distribution Table $(SNDT)$ converts the Robust Z-Score results and the percent coverage.

The percentage representing the area below the curve could be adopted to evaluate the anomaly. The idea leverages the assumption that x_o is not an anomaly if it is in a dense area. Thus, the $LRD(x_o)$ is equal to or higher than the median of the distribution of the LRDs of the other points of the cluster C_{x_o}. Therefore, the value obtained by $RobustZ-Score(x_o)$ is high, and thus the covered area is high too.

Hence, the more area is covered, the higher the probability that x_o is not an anomaly. Alternatively, vice versa, the more uncovered area there is, the more the probability that the point x_o is an anomaly. Therefore, the formula is:

$$DensityEvaluation = 1 - SNDT(Robust\ Z\text{-}Score(LRD(x_o))) \tag{5.6}$$

5.3 Algorithm

The algorithm proposed for discovering anomalies in time series data is based on three main phases: Learning Phase, Estimation Parameter Phase, and Evaluation Phase. Figure 5.7 shows the entire pipeline of the algorithm.

5.3.1 Learning Phase

The cold start is the main problem of anomaly detection algorithms on streaming data[55, 122, 105]. It concerns the impossibility of deducting any information because the system has not yet gathered sufficient data. In literature, the cold start is solved by providing an initial Learning Phase. This phase helps the system to gather data for estimating some

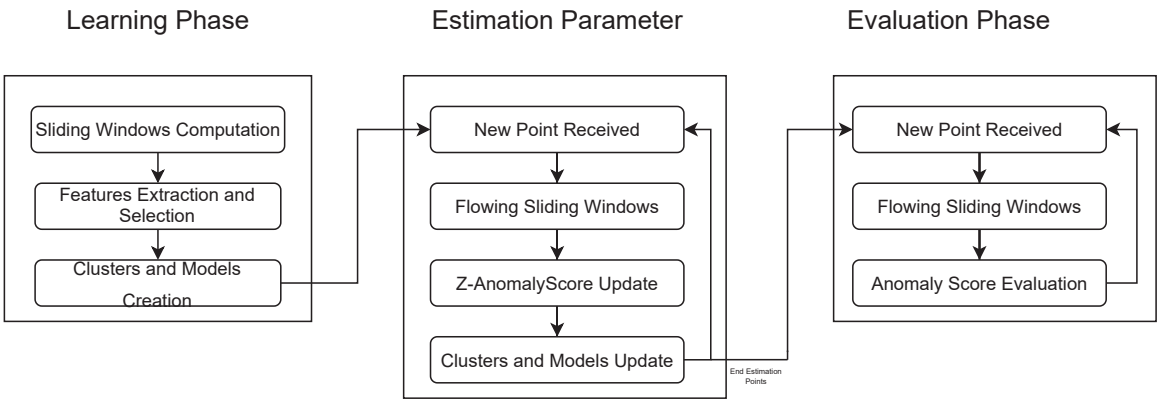


Fig. 5.7 AnomalyFeats Pipeline

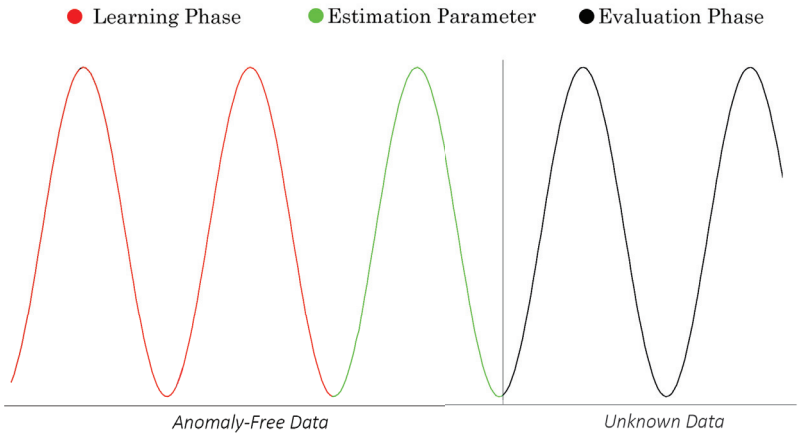


Fig. 5.8 Division between Anomaly-Free and Unknown Data

characteristics of the data domain. The data provided during this phase are anomaly-free for not falsifying the information detected, as shown in Figure 5.8.

For our purpose, the Learning Phase permits us to discover which features can represent the domain of the data provided in input and which is the peculiarity of the features selected. As shown in previous work, feature extraction is an operation made on a fixed number of points in the temporal series. In streaming data, the points are obtained continuously, and it is impossible to define a priori a fixed number of points. Therefore, our algorithm solves this problem by adopting a model that permits looking after a subset of all the points, i.e., the sliding window. The sliding window is a structure with a fixed number of elements t that flow among the series when a new element arrives and rejects the oldest point of the series. Formally, once obtained x_{lp}, the last point received in a streaming way during the Learning

Last Point	*mean*	*trend_stderr*	*variance*	*peaks*	*quantile*	*trend_rvalue*
x_{lp}	51.3	3.51	788.56	8	57	-0.94

Table 5.1 Example of a subset of features extracted on the Sliding Windows of length t on the last point x_{lp}

Last Point	*mean*	*trend_stderr*	*variance*	*peaks*	*quantile*	*trend_rvalue*
x_{lp}	51.3	3.51	788.56	8	57	-0.94
x_{lp-1}	50.2	3.91	745.96	4	51	-0.91
x_{lp-2}	54.8	4.01	792.05	6	56	-0.89
...	...	...	...	...	...	...

(a) Matrix of features extracted on sw_t

Last Point	*mean*	*trend_stderr*	*variance*	*peaks*	*quantile*	*trend_rvalue*
x_{lp}	38.3	10.41	491.66	25	91	-4.94
x_{lp-1}	41.7	12.95	422.45	30	88	-5.84
x_{lp-2}	48.3	15.21	477.96	21	98	-3.99
...	...	...	...	...	...	...

(b) Matrix of features extracted on sw_s

Table 5.2 Example of the features extracted with two different size of Sliding Windows

Phase, we define the sliding windows sw_t, a data structure that contains the last t points of the streaming:

$$sw_t = [x_{lp}, x_{lp-1}, ...x_{lp-t}]$$

The size of the sliding window is a delicate parameter to choose. Indeed, big sliding windows can extract efficient time series but can approximate small shifts in the series that can be anomalies. Conversely, small sliding windows can reveal these small shifts, but the features could not be reliable. Therefore, our algorithm proposes adopting multiple sliding window sizes to balance the previously exposed problems. Formally, we define SW as the set of all the sliding windows size i:

$$SW = \{sw_i : i \in Z^+\}$$

Hence, adopting multiple sliding windows permits observing different peculiarities of the same signal and obtaining different insights for evaluating the quality of the time series point.

Moreover, the sliding windows permit obtaining a fixed number of points where it is possible to extract the features.

Indeed, selecting a sliding window $sw_t \in SW$, where x_{lp} is the last point received in the streaming data, the features extraction algorithm extracts all the possible features on t points. Therefore, we define as $F_{t,x_{lp}}$, the features extracted on the sliding window with size t when the point x_{lp} is received. Table 5.1 shows an example of an array of a subset of extracted features.

When a new point in the series arrives, this extraction operation is repeated on each sliding window $sw_i \in SW$. When all the points of the Learning Phase are evaluated, the features extracted from each sliding window are combined, obtaining a matrix of features for each sliding window $sw_i \in SW$, as shown in Table 5.2.

Features Selected	Weight
trend_stderr	0.55
variance	0.35
...	...

(a) Features Selected SF_t on sw_t

Features Selected	Weight
mean	0.65
variance	0.15
...	...

(b) Features Selected SF_s on sw_s

Table 5.3 Example of the features selected for two different sizes of Sliding Windows

(a) Clusters $CSF_{t,j}$ of the SF_t

(b) Clusters $CSF_{s,j}$ of the SF_s

Fig. 5.9 Clusters creation for each selected feature.

The number of features extracted from each sliding window can be highly impactful in terms of memory. Moreover, not all the features are meaningful for representing the input signal. Therefore, an analysis of extracted features is needed to select the most prominent features that better represent the signal. In our algorithm, we adopt the PFA for selecting the best features of the signals. In addition to the property of feature selection, the PFA permits evaluating the importance of each chosen feature. Therefore, we define as SF_t the set of features extracted from the sliding window with length t, where each feature has a weight derived from the PFA that is adopted in the computation of the anomaly. The features selected from each sliding window can be different, as shown in the Tables 5.3. We also keep the weight computed by the PFA for each feature selected.

In Section 5.2, the clusters are employed to evaluate the points quality during the Evaluation Phase. Our algorithm computes these clusters on the features selected by the PFA for each sliding window size. Indeed, we define $CSF_{i,j}$, the cluster dedicated to the features selected j for the sliding window with size i, as shown in Figures 5.9. During the creation of the clusters, the algorithm computes the Local Reachability Distances used in the Density Evaluation in Formula 5.6.

The incremental neural networks are a machine learning technique adopted for improving the system's quality. The potentiality of these models derives from their ability to train the models online, i.e., the models are updated each time new data arrives, as shown in Figure 5.10.

The incremental neural networks adopted are based on regression models. These regression models forecast new values based on the values previously seen. We define $MSF_{i,j}$ the regression model M trained on the values of the feature selected j in the sliding windows with size i. Figure 5.11 shows an example of a Incremental Neural Network based a Regression Model for the $trend\_stderr$ feature.

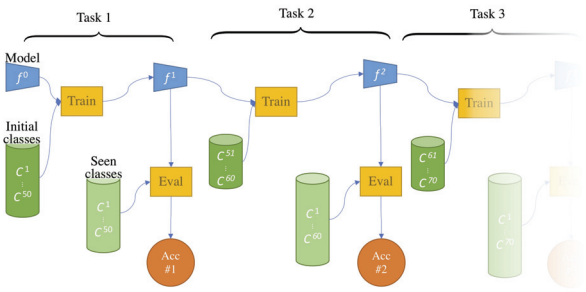


Fig. 5.10 Example of Incremental Neural Network

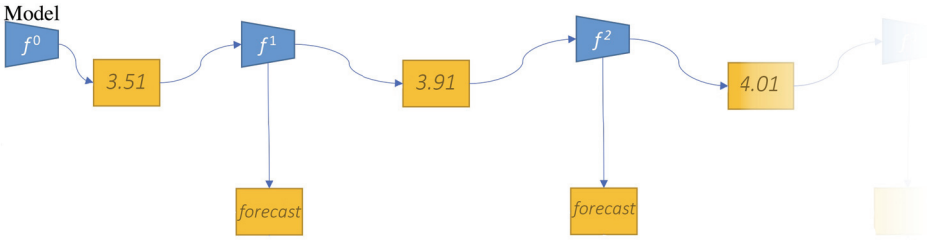


Fig. 5.11 Incremental Model $MSF_{t,i}$

5.3.2 Estimation Parameter

During this phase, some parameters for evaluating the anomaly are calibrated. The idea is to compute the formulas presented in Section 5.2 on not anomalous data to detect the 'expected' value of the formula among usual points. The points adopted during this phase are defined as EP and are indicated in red in Figure 5.8. These points belong to the Anomaly-Free data and are adopted by the estimation parameter for calibrating the parameters. In detail, each point $x_{ep} \in EP$ of the series is given in input to the algorithm during this phase. The system moves all the sliding windows for each point and evaluates the sliding windows by applying the formula presented in Section 5.2. Therefore, for each sliding window of size t, we obtain an $AnomalyScore_t(x_{ep})$. At the end of all the points in EP, the algorithm produces a different distribution of AnomalyScore for each sliding window size. In each of these distributions, the values generated can be different for each sliding window, even if they are generated on the same free-anomaly data, as shown in Table 5.4.

Therefore, we opted for creating a single distribution that tries to merge all the distributions into one. We have adopted the Z-Score to normalize the distributions for obtaining similar values in the case of no anomaly data. Indeed, as shown in Formula 5.7, the Z-Score compares each anomaly score value with the average of the anomaly score distribution and divides the value by the standard distribution.

Sliding Window	Anomaly Score
$[x_{ep}, x_{ep-1}, ...x_{ep-t}]$	0.52
$[x_{ep-1}, x_{ep-2}, ...x_{ep-t-1}]$	0.59
...	...

(a) Anomaly Score Distribution on sw_t

Sliding Window	Anomaly Score
$[x_{ep}, x_{ep-1}, ...x_{ep-s}]$	0.22
$[x_{ep-1}, x_{ep-2}, ...x_{ep-s-1}]$	0.29
...	...

(b) Anomaly Score Distribution on sw_s

Table 5.4 Example of the Anomaly Score distribution of the two different size of Sliding Windows

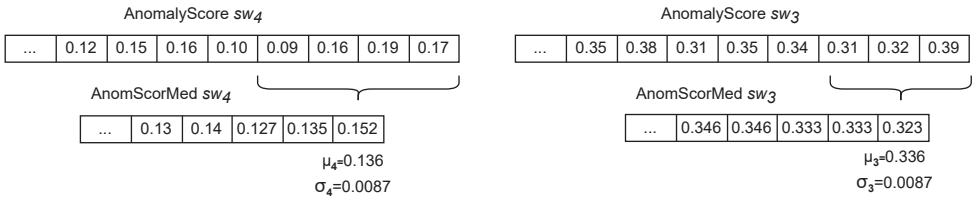


Fig. 5.12 Computation of the *AnomScoreMean*, μ and σ on sliding windows 3 and 4.

$$Z - Score_t(x_{ep}) = \frac{AnomScore_t(x_{ep}) - \mu_t}{\sigma_i} \tag{5.7}$$

In the formula, μ_i and the σ_i are respectively the average and the standard distribution of the anomaly score of the sliding window with size i.

This solution permits obtaining Z-Score results close to 0 when the new anomaly score is similar to the distribution average and results far from 0 when the new anomaly score is far from the average. The Z-Score can assume a range of elements between $-inf$ and $+inf$; therefore, as introduced for the Robust Z-Score in section 5.2, we use the Standard Normal Distribution Table to reduce the range between 0 and 1. Hence, we define Z-AnomalyScore as:

$$Z - AnomalyScore(x_{ep}) = \frac{\sum_i^M Z - Score_i(x_{ep})}{\|M\|} \tag{5.8}$$

M is the set of all the sizes of the sliding windows. This formula can be improved to obtain better results. Indeed, the $AnomalyScore(x_{ep})$ can be replaced by computing the average of the last t anomaly score, where t is the size of the sliding window[2]. Formally, we define $AnomScoreMean_t(x_{ep})$as:

$$AnomScoreMean_t(x_{ep}) = \frac{\sum_{i=0}^{t} AnomScore(x_{ep-i})}{t} \tag{5.9}$$

Where t is the sliding window size and $AnomalyScore(x_{ep-i})$ is the anomaly score obtained on the i point before the x_{ep}. Figure 5.12 shows an example of how the $AnomScoreMean_t$

...	x_{ep-4}	x_{ep-3}	x_{ep-2}	x_{ep-1}	x_{ep}
...	0.5600	0.7759	0.2577	0.4096	0.5173

Table 5.5 Distribution of the $Z-AnomalyScore$

is computed on two different sliding windows. Therefore, $Z-AnomalyScore(x_{ep})$ is defined as:

$$Z-AnomalyScore(x_{ep}) = \frac{\sum_i^M SNTD(\frac{AnomScoreMean_i(x_{ep})-\mu_i}{\sigma_i})}{\|M\|} \tag{5.10}$$

In summary, this formula permits merging the anomaly scores obtained on M different sliding windows in one value for each value x_{ep} obtained. Equation 5.11 shows an example of application of the Z-AnomalyScore on the last value computed in Figure 5.12.

$$Z-AnomalyScore(x_{ep}) = \frac{\frac{0.152-0.136}{0.0087}+\frac{0.323-0.336}{0.0087}}{2} \underset{\text{because of } SNDT}{=} \frac{0.9670+0.0675}{2} = 0.5173 \tag{5.11}$$

At the end of the points in EP, we have a set of values that defines the distribution of the $Z-AnomalyScore$ on anomaly-free data. This distribution is a crucial parameter for revealing the anomaly for evaluating future points. Indeed, $Z-AnomalyScores$ that are not in line with the distribution computed during the estimation parameter can reveal an anomaly.

In detail, computed the distribution of the $Z-AnomalyScore$ as shown in Table 5.5, our algorithm defines the center of the distribution as the maximum threshold for defining not anomalous a new point in the evaluation phase. This assumption is based on the construction of the Formula 5.1. Indeed, the formula returns the anomaly score of a point based on the previous points obtained and the learning phase. During the estimation parameter phase, the formula computes the *AnomalyScore* on not anomalous data. Therefore, the distribution center represents the most common $Z-AnomalyScore$ for the not-anomalous data. Hence, if the $Z-AnomalyScore$ of a point in the evaluation phase is bigger than the distribution center, it is bigger than the most common *AnomalyScore* of the not-anomalous data, and it could be an anomaly. Formally, we define $Z-AnomalyTreshold$ as the maximum threshold for considering points not-anomalous. For example, the $Z-AnomalyTreshold$ on the distribution of the $Z-AnomalyScore$ in Table 5.5 is around *0.50*.

Unfortunately, the definition of this threshold for declaring a point anomaly is a very stringent condition. As expressed in the last paragraphs, the computation of this threshold is based on the distribution of the *Z-AnomalyScore*. Therefore, a not anomalous point can

...	x_{ep-4}	x_{ep-3}	x_{ep-2}	x_{ep-1}	x_{ep}
...	0.6371	0.9466	0.0782	0.2985	0.5402

Table 5.6 Distribution Difference of the $Z-AnomalyScore$

differ by some value from the *Z-AnomalyThreshold* declared. Thus, it appears helpful to compute a boundary of tolerance for accepting normal point that differs from the *Z-AnomalyThreshold*. Our algorithm defines the tolerance by adopting the difference in terms of percentage between the *Z-AnomalyScore* and *Z-AnomalyThrehsold*. Therefore, we define Formula 5.12 for computing the difference in percentage between these two values.

$$percentageAnomaly(x) = \frac{|Z-AnomalyScore(x) - Z-AnomalyThreshold|}{\frac{Z-AnomalyScore(x)-Z-AnomalyThreshold}{2}} \tag{5.12}$$

Once the difference is computed, we provide two methods for expressing the boundary. In the first, the user can decide the maximum difference in terms of the percentage the system can accept. In the second method, we compute the percentage difference, leveraging the distribution of the *Z-AnomalyScore*. The idea is to observe how much the *Z-AnomalyScores* computed during the Estimation Parameter phase are far from the center of the distribution. The *Z-Score* perfectly suits this purpose. Our algorithm computes the *Z-Score* function by adopting the *Z-AnomalyScore* of each EP point and the average and the standard deviation of the *Z-AnomalyScore* distribution. Table 5.6 shows an example of the distribution differences by adopting the *Z-AnomalyScore* distribution in Table 5.5. All the results are computed by adopting a $\mu = 0.50$ and $\sigma = 0.171$ and already converted in the range between 0 and 1, adopting the Standard Normal Distribution Table.

Then, we compute the average of these values that represents the percentage of how much the *Z-AnomalyScore* can differ from the *Z-AnomalyThreshold*. In the case of the distribution of the Table 5.5, the difference accepted is up to 50% more than the *Z-AnomalyThreshold*.

Finally, clusters and models created and trained during the Learning Phase are updated during the Estimation Parameter with the values of the features extracted.

5.3.3 Evaluation Phase

In this phase, the algorithm evaluates the quality of the points on the unknown data. The process of anomaly recognition during the Evaluation Phase is very similar to the Estimation parameter Phase. Indeed, at the reception of a new point x, the algorithm computes the $Z-AnomalyScore(x)$ to reveal the quality of this point. In case the *Z-AnomalyScore* of x does not overpass the *Z-AnomalyThreshold*, the algorithm computes the same step seen

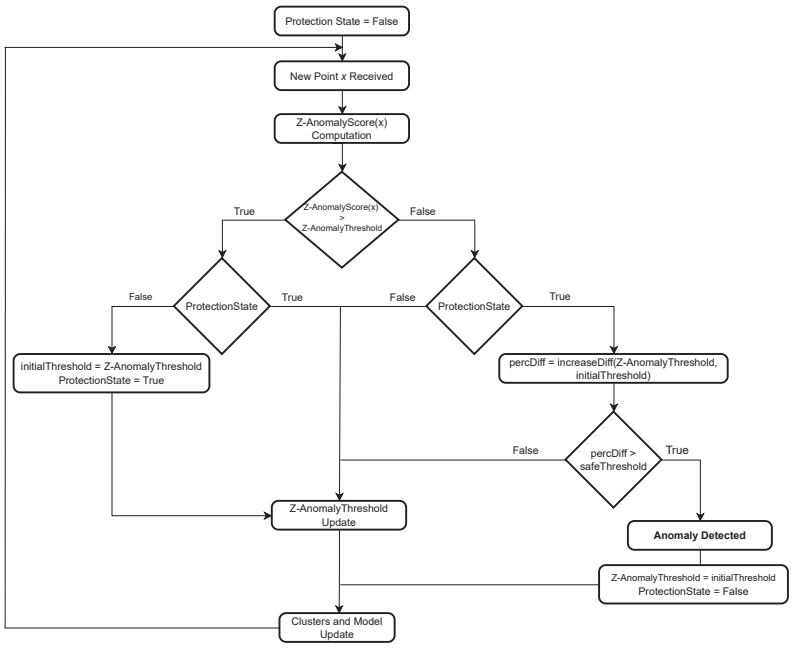


Fig. 5.13 Evaluation Phase Flowchart

during the Evaluation phase, i.e., updating the *Z-AnomalyTreshold* and calibration of the tolerance percentage.

Conversely, if the *Z-AnomalyScore* is higher than the *Z-AnomalyTreshold*, the algorithm actives a 'Protection State'. During this state, the algorithm saves the *Z-AnomalyThreshold* obtained before the state of protection of the system and continues accepting new points, computing the *Z-AnomalyScore*, and updating the *Z-AnomalyThreshold*. During this protection state, the value of the *Z-AnomalyTreshold* tends to augment. The algorithm stops the protection state when, during the process of accepting the points, the *Z-AnomalyScore* obtains a value minor of the *Z-AnomalyThreshold*, i.e., when the *Z-AnomalyTreshold* stops to increase. This *Z-AnomalyThreshold*, obtained during the protection state, will be compared with the *Z-AnomalyTreshold* obtained before the protection state. Suppose the difference expressed in terms of percentage adopting Formula 5.12 is higher than the threshold computed. In that case, all the points received during that phase are considered anomalies, and the *Z-AnomalyThreshold* will be restored. Conversely, if the percentage is minor, then the points are not considered anomalies, and the parameters saved before the protection state are updated after these new points. The incremental neural network models and the clusters are constantly updated during the protection state. The motivation is explained in the next section. Figure 5.13 shows the entire process of the Evaluation Phase.

5.3.4 Streaming Anomaly Detection Properties

In Section 5.1, we have introduced four main properties that an anomaly detection algorithm on streaming data has to respect. In this section, we demonstrate that our algorithm respects all these requests.

Transient: Our algorithm respects this property because the process immediately activates a protection state when an anomaly or a subset of anomalies is recognized and notifies the user of these anomalies when the points return in a recognized state.

Infinite: Our algorithm respects this property because all the averages and standard deviations adopted during the process are computed through the Welford method. This technique does not need to keep in memory all the previous values. Therefore, it has an impact constant on the memory. The Clusters and the Incremental Neural Networks can cause concerns about the high impact on memory. Our algorithm solves these problems by adopting optimization systems of the clusters that permit removing old and redundant points from the agglomerations. Furthermore, the Incremental Neural Networks adopted by our system automatically discard data constantly seen from the network.

Arrival Rate: Our algorithm respects this property because when an anomaly is received, the computation of the AnomalyScore is immediate. Therefore, the algorithm processes the data point before the reception of the other points.

Concept Drift: Our algorithm respects this property by adopting the Clusters and the Neural Network. The clusters keep in memory the most representative values of features that usually are the most repeated during the reception of the points. When an anomaly is detected, the features' value is presumably in a low-density cluster area because the point was never seen before. By definition of anomaly, this area should remain lowly dense because the anomaly is a rare event. If the points in this area augment, this means that the original signal is changing. Therefore, keeping in memory this low-density area for enough time, the algorithm can reveal a concept drift. As a consequence, in case some high-density areas are not even more populated when new points are received, through the process of optimization cluster, those areas are slowly reduced until they are completely deleted. The Incremental Neural Network models respect the concept drift property by construction. Indeed, based on their constant training, they automatically adapt to the changing of the signal.

5.4 Experiments

In this section, we introduce the first results obtained by our algorithm among some datasets provided by Numenta Anomaly Benchmark (NAB)[2]. NAB is a benchmarking system that allows the evaluation of algorithms for anomaly detection in streaming. It comprises over

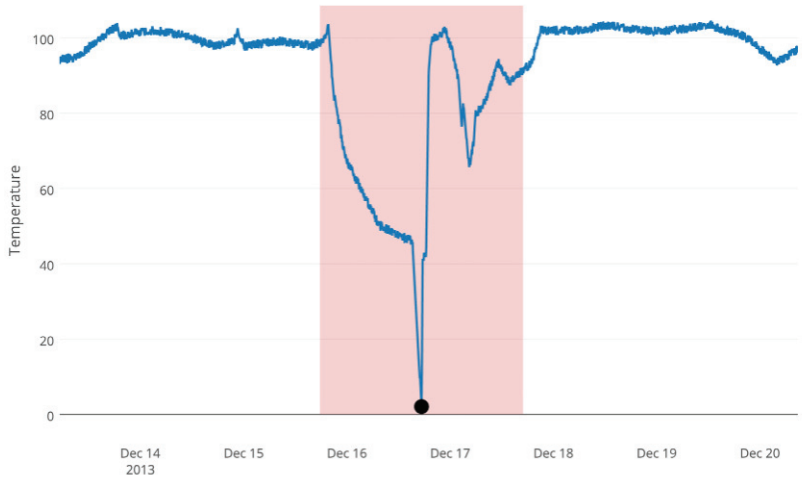


Fig. 5.14 Example of Anomaly Window in a NAB Scoring.

50 labeled real-world and artificial time series data files and a novel scoring mechanism for real-time applications. The scoring mechanism provided by NAB proposes a set of rules to determine the overall quality of anomaly detection by following the ideal requirements for a real-world anomaly detector:

1. Detects all anomalies present in the streaming data
2. Detects anomalies as soon as possible
3. Triggers no false alarms (no false positives)
4. Works with real-time data (no look ahead)
5. Fully automated across all datasets (any data specific parameter tuning must be done online without human intervention)

These rules are implemented by adopting three key elements: the anomaly windows, the scoring function, and application profiles. The anomaly windows are a range of data points centered around the anomaly labels that allow awarding the early detection of the algorithms. Figure 5.14 shows an example of how the windows are created. The range of data points adopted for creating the windows is equal to 10% of the length of a data file, divided by the number of anomalies in the given file. Once created the anomaly windows, a scoring function uses these windows to identify and weight true positives, false positives, and false negatives. The scoring function is based on a scaled sigmoidal function, which defines the weight of individual detections given the anomaly window and the relative position of each detection. This function gives higher positive scores to true positive detections earlier in a window, and

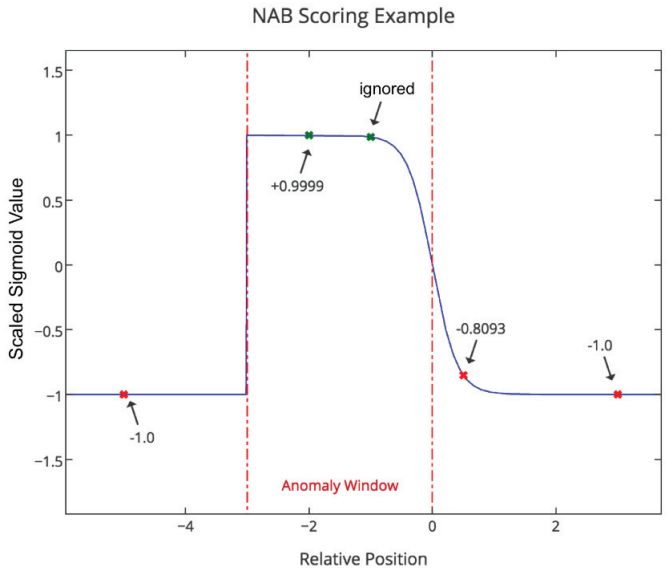


Fig. 5.15 Scoring example for a sample anomaly window, where the values represent the scaled sigmoid function, the second term in Eq. (13).

	Standard Profile	Reward Low FPs	Reward Low FNs
Atp	1	1	1
Atn	1	1	1
Afp	0.11	0.22	0.11
Afn	1	1	2

Table 5.7 Application Profiles of NAB

detections slightly after the window contribute less to negative scores than detections well after the window, as shown in Figure 5.15.

The sole purpose of this function is to reveal the importance of the position where the anomaly is detected, indicating if the point is a true positive, true negative, or false negative. Different applications may emphasize the relative importance of true positives, false negatives, and false positives. Therefore, NAB introduces the notion of application profiles. For TPs, FPs, FNs, and TNs, NAB applies different relative weights associated with each profile to obtain a separate score per profile. NAB includes three application profiles: standard, reward low FPs and reward low FNs. In Table 5.7 are indicated the weights of every single configuration.

Equation 13 defines the scaled sigmoidal function. Let A be the application profile under consideration, with A_{TP}, A_{FP}, A_{TN}, and A_{FN} the corresponding weights for true positives, false positives, etc. These weights are bounded $0 \leq A_{TP}, A_{TN} \leq 1$, and $-1 \leq A_{FP}, A_{FN} \leq 0$.

The y in the equation represents the relative position of the detection within the anomaly window.

$$\sigma^A(y) = (A_{TP} - A_{FP})(\frac{1}{1+e^{5y}}) - 1 \tag{5.13}$$

In Eq. (13), the parameters are set such that the right end of the window evaluates to $\sigma(y) = 0.0 = 0$, yielding a max and min of A_{TP} and A_{FP} respectively. Every detection outside the window is counted as a false positive and given a scaled negative score relative to the preceding window. The number of windows with zero detections in the data file is the number of false negatives A_{FN}, represented by fd.

The raw score for a data file Y_d is the sum of the scores from individual detections plus the impact of missing any windows. Eq. (14) accumulates the weighted score for each true positive and false positive and detriments the total score with a weighted count of all the false negatives.

$$S_d^A = (\sum_{y \in Y_d} \sigma^A(y)) + A_{FN} fd \tag{5.14}$$

The benchmark raw score for a given algorithm is the sum of the raw scores over all the tested datasets D.

$$S^A = \sum_{d \in D} S_d^A \tag{5.15}$$

The final reported NAB scores are normalized such that the perfect score is 100, and a "null" detector defines the baseline of 0. However, this is not the minimum; negative scores are possible. Equation 16 shows the normalized reported score of NAB.

$$S_{NAB}^A = \frac{S^A - S_{null}^A}{S_{perfect}^A - S_{null}^A} \tag{5.16}$$

We have adopted only six datasets in our experiments to test our algorithm. The final reported score on these datasets equals 16%, based on the standard application profile. The result does not outperform state-of-the-art, but this algorithm is still under construction. Therefore, we aim to simplify and improve the algorithm's pipeline for better results.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

This research aimed to design a novel and effective method for solving the most interesting data mining problems researched in the last years in the time series domain: Clustering and Anomaly Detection. In this thesis, we solve these problems by leveraging statistical features extracted from the time series. The novelty of our work consists in automatically selecting the most appropriate statistical features based on the dataset provided as input. Not all the features have the same quality, and choosing a subset of high-quality features for each dataset is beneficial for the clustering step. This characteristic is fruitful in several real-life data sciences and data analytics pipelines. Indeed, the time series features are interpretable by humans, thus leading to a more transparent and human-centric clustering process, which is helpful for the user interested in in-depth analysis. To the best of our knowledge, our solutions are the first feature-based semi-supervised clustering and anomaly detection frameworks with these key properties. Moreover, based on the results obtained, adopting these features permits a substantial complexity reduction expressed in memory and time. Our work on time series clustering shows that there is no one-size-fits-all solution regarding the set of features to use. The results produced by our two open source[1][2] algorithms show that the features are an interesting tool for obtaining high-quality, fast, and interpretable solutions.

[1] https://github.com/protti/FeatTS

[2] https://github.com/softlab-unimore/time2feat

6.2 Future Work

Here, we give some perspectives on our current contributions. FeatTS could be improved by rendering the entire pipeline unsupervised instead of the current semi-supervised approach. This requires non-trivial extensions in order to be able to cluster the time series without loss of performance. Another improvement would be to dynamically choose the threshold for graph creation based on the processed features. Finally, the community detection algorithm's weights could be combined with the features' relevance degrees. Time2Feat could be improved by creating a new feature extraction method that permits extracting features by providing in input all the signals of the multivariate series and not only a couple of them. Finally, AnomalyFeat could be improved by diminishing the number of points to keep in memory to reduce computational time. Our work is only the starting point for new fascinating solutions with the features. Many other Data Mining problems using time series data can be solved by adopting this technique. Other kinds of data allow extracting features. An example are graph-shaped data in which nodes and properties have dynamic components, such as sequential values. Such hybrid networks have inherent semantics and can still benefit from feature-based data mining tasks. Finally, the solutions proposed in this thesis provide food for thought for many other data science problems.

References

[1] Aggarwal, C. C. (2017). An introduction to outlier analysis. In *Outlier analysis*, pages 1–34. Springer.

[2] Ahmad, S., Lavin, A., Purdy, S., and Agha, Z. (2017). Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147.

[3] Albawi, S., Mohammed, T. A., and Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee.

[4] Altmann, E. G., Hallerberg, S., and Kantz, H. (2006). Reactions to extreme events: Moving threshold model. *Physica A: Statistical Mechanics and its Applications*, 364:435–444.

[5] Bagnall, A., Lines, J., Hills, J., and Bostrom, A. (2015). Time-series classification with cote: the collective of transformation-based ensembles. *IEEE Transactions on Knowledge and Data Engineering*, 27(9):2522–2535.

[6] Bagnall, A., Ratanamahatana, C., Keogh, E., Lonardi, S., Janacek, G., et al. (2006). A bit level representation for time series data mining with shape based similarity. *Data mining and knowledge discovery*, 13(1):11–40.

[7] Bagnall, A. J., Dau, H. A., Lines, J., Flynn, M., Large, J., Bostrom, A., Southam, P., and Keogh, E. J. (2018). The UEA multivariate time series classification archive, 2018. *CoRR*, abs/1811.00075.

[8] Barandas, M., Folgado, D., Fernandes, L., Santos, S., Abreu, M., Bota, P., Liu, H., Schultz, T., and Gamboa, H. (2020). Tsfel: Time series feature extraction library. *SoftwareX*, 11:100456.

[9] Basu, S., Banerjee, A., and Mooney, R. (2002). Semi-supervised clustering by seeding. In *In Proceedings of 19th International Conference on Machine Learning (ICML-2002*. Citeseer.

[10] Baydogan, M. G. and Runger, G. (2016). Time series representation and similarity based on local autopatterns. *Data Mining and Knowledge Discovery*, 30(2):476–509.

[11] Bebis, G. and Georgiopoulos, M. (1994). Feed-forward neural networks. *IEEE Potentials*, 13(4):27–31.

[12] Begum, N., Ulanova, L., Wang, J., and Keogh, E. (2015). Accelerating dynamic time warping clustering with a novel admissible pruning strategy. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 49–58.

[13] Benjamini, Y. and Yekutieli, D. (2001). The control of the false discovery rate in multiple testing under dependency. *Annals of statistics*, pages 1165–1188.

[14] Berndt, D. J. and Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, USA:.

[15] Borges, J. a. B., Ramos, H. S., and Loureiro, A. A. F. (2022). A classification strategy for internet of things data based on the class separability analysis of time series dynamics. *ACM Trans. Internet Things*.

[16] Bostrom, A. and Bagnall, A. (2015). Binary shapelet transform for multiclass time series classification. In *International conference on big data analytics and knowledge discovery*, pages 257–269. Springer.

[17] Box, G. E., Jenkins, G. M., Reinsel, G. C., and Ljung, G. M. (2015). *Time series analysis: forecasting and control*. John Wiley & Sons.

[18] Braei, M. and Wagner, S. (2020). Anomaly detection in univariate time-series: A survey on the state-of-the-art. *arXiv preprint arXiv:2004.00433*.

[19] Breaban, M. and Luchian, H. (2011). A unifying criterion for unsupervised clustering and feature selection. *Pattern Recognition*, 44(4):854–865.

[20] Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000a). Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104.

[21] Breunig, M. M., Kriegel, H.-P., Ng, R. T., and Sander, J. (2000b). Lof: Identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, page 93–104, New York, NY, USA. Association for Computing Machinery.

[22] Buono, P., Aris, A., Plaisant, C., Khella, A., and Shneiderman, B. (2005). Interactive pattern search in time series. *Visualization and Data Analysis 2005*, 5669:175–186.

[23] Campello, R. J., Kröger, P., Sander, J., and Zimek, A. (2020). Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(2):e1343.

[24] Carlis, J. V. and Konstan, J. A. (1998). Interactive visualization of serial periodic data. In *Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 29–38.

[25] Chakrabarti, K., Keogh, E., Mehrotra, S., and Pazzani, M. (2002). Locally adaptive dimensionality reduction for indexing large time series databases. *ACM Transactions on Database Systems (TODS)*, 27(2):188–228.

[26] Chandola, V., Banerjee, A., and Kumar, V. (2009). Survey of anomaly detection. *ACM Computing Survey (CSUR)*, 41(3):1–72.

[27] Chauhan, S. and Vig, L. (2015). Anomaly detection in ecg time signals via deep long short-term memory networks. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–7. IEEE.

[28] Chen, M., Shi, X., Zhang, Y., Wu, D., and Guizani, M. (2017). Deep feature learning for medical image analysis with convolutional autoencoder neural network. *IEEE Transactions on Big Data*, 7(4):750–758.

[29] Chen, X.-w. and Jeong, J. C. (2007). Enhanced recursive feature elimination. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 429–435. IEEE.

[30] Christ, M., Braun, N., Neuffer, J., and Kempa-Liehr, A. W. (2018). Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, 307:72–77.

[31] Cook, A. A., Mısırlı, G., and Fan, Z. (2019). Anomaly detection for iot time-series data: A survey. *IEEE Internet of Things Journal*, 7(7):6481–6494.

[32] Dang, T. N. and Wilkinson, L. (2013). Timeexplorer: Similarity search time series by their signatures. In *International Symposium on Visual Computing*, pages 280–289. Springer.

[33] Dau, H. A., Bagnall, A., Kamgar, K., Yeh, C.-C. M., Zhu, Y., Gharghabi, S., Ratanamahatana, C. A., and Keogh, E. (2018). The ucr time series archive.

[34] Desgraupes, B. (2013). Clustering indices. *University of Paris Ouest-Lab Modal'X*, 1:34.

[35] Dhanabal, S. and Chandramathi, S. (2011). A review of various k-nearest neighbor query processing techniques. *International Journal of Computer Applications*, 31(7):14–22.

[36] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. (2008). Querying and mining of time series data: experimental comparison of representations and distance measures. *Proceedings of the VLDB Endowment*, 1(2):1542–1552.

[37] Fuchs, E., Gruber, T., Pree, H., and Sick, B. (2010). Temporal data mining using shape space representations of time series. *Neurocomputing*, 74(1-3):379–393.

[38] Fulcher, B. D. and Jones, N. S. (2017). hctsa: A computational framework for automated time-series phenotyping using massive feature extraction. *Cell systems*, 5(5):527–531.

[39] Giannoni, F., Mancini, M., and Marinelli, F. (2018). Anomaly detection models for iot time series data. *arXiv preprint arXiv:1812.00890*.

[40] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press.

[41] Grabocka, J., Schilling, N., Wistuba, M., and Schmidt-Thieme, L. (2014). Learning time-series shapelets. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 392–401.

[42] Guijo-Rubio, D., Durán-Rosal, A. M., Gutiérrez, P. A., Troncoso, A., and Hervás-Martínez, C. (2020). Time-series clustering based on the characterization of segment typologies. *IEEE transactions on cybernetics*, 51(11):5409–5422.

[43] Guo, T., Xu, Z., Yao, X., Chen, H., Aberer, K., and Funaya, K. (2016). Robust online time series prediction with recurrent neural networks. In *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 816–825. Ieee.

[44] Guo, Y., Liao, W., Wang, Q., Yu, L., Ji, T., and Li, P. (2018). Multidimensional time series anomaly detection: A gru-based gaussian mixture variational autoencoder approach. In *Asian Conference on Machine Learning*, pages 97–112. PMLR.

[45] Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using networkx. In Varoquaux, G., Vaught, T., and Millman, J., editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA.

[46] Hamilton, J. D. (2020). *Time series analysis*. Princeton university press.

[47] Hautamaki, V., Nykanen, P., and Franti, P. (2008). Time-series clustering by approximate prototypes. In *2008 19th International conference on pattern recognition*, pages 1–4. IEEE.

[48] He, L., Agard, B., and Trépanier, M. (2020). A classification of public transit users with smart card data based on time series distance metrics and a hierarchical clustering method. *Transportmetrica A: Transport Science*, 16(1):56–75.

[49] Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier.

[50] Henry, M. and Judge, G. (2019). Permutation entropy and information recovery in nonlinear dynamic economic time series. *Econometrics*, 7(1):10.

[51] Hills, J., Lines, J., Baranauskas, E., Mapp, J., and Bagnall, A. (2014). Classification of time series by shapelet transformation. *Data mining and knowledge discovery*, 28(4):851–881.

[52] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[53] Hollingsworth, K., Rouse, K., Cho, J., Harris, A., Sartipi, M., Sozer, S., and Enevoldson, B. (2018). Energy anomaly detection with forecasting and deep learning. In *2018 IEEE international conference on big data (Big Data)*, pages 4921–4925. IEEE.

[54] Horng, S.-J., Su, M.-Y., Chen, Y.-H., Kao, T.-W., Chen, R.-J., Lai, J.-L., and Perkasa, C. D. (2011). A novel intrusion detection system based on hierarchical clustering and support vector machines. *Expert systems with Applications*, 38(1):306–313.

[55] Huang, T., Chen, P., and Li, R. (2022). A semi-supervised vae based active anomaly detection framework in multivariate time series for online systems. In *Proceedings of the ACM Web Conference 2022*, pages 1797–1806.

[56] Huang, X., Ye, Y., Xiong, L., Lau, R. Y., Jiang, N., and Wang, S. (2016). Time series k-means: A new k-means type smooth subspace clustering for time series data. *Information Sciences*, 367:1–13.

[57] Idé, T. (2006). Why does subsequence time-series clustering produce sine waves? In *european conference on principles of data mining and knowledge discovery*, pages 211–222. Springer.

[58] Ienco, D. and Interdonato, R. (2020). Deep Multivariate Time Series Embedding Clustering via Attentive-Gated Autoencoder. In *PAKDD 2020 - 24th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Advances in Knowledge Discovery and Data Mining, Singapore, Singapore.

[59] Inoue, A. (2008). Ar and ma representation of partial autocorrelation functions, with applications. *Probability theory and related fields*, 140(3):523–551.

[60] Ismail Fawaz, H., Forestier, G., Weber, J., Idoumghar, L., and Muller, P.-A. (2019). Deep learning for time series classification: a review. *Data mining and knowledge discovery*, 33(4):917–963.

[61] Iwana, B. K. and Uchida, S. (2021). An empirical survey of data augmentation for time series classification with neural networks. *Plos one*, 16(7):e0254841.

[62] Kadri, F., Harrou, F., Chaabane, S., Sun, Y., and Tahon, C. (2016). Seasonal arma-based spc charts for anomaly detection: Application to emergency department systems. *Neurocomputing*, 173:2102–2114.

[63] Kang, Y., Hyndman, R. J., and Li, F. (2020). Gratis: Generating time series with diverse and controllable characteristics. *Statistical Analysis and Data Mining: The ASA Data Science Journal*.

[64] Karimian, S. H., Kelarestaghi, M., and Hashemi, S. (2012). I-inclof: improved incremental local outlier detection for data streams. In *The 16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP 2012)*, pages 023–028. IEEE.

[65] Kashyap, S. and Karras, P. (2011). Scalable knn search on vertically stored time series. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1334–1342.

[66] Kayabol, K. (2019). Approximate sparse multinomial logistic regression for classification. *IEEE transactions on pattern analysis and machine intelligence*, 42(2):490–493.

[67] Keogh, E., Chakrabarti, K., Pazzani, M., and Mehrotra, S. (2001). Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and information Systems*, 3(3):263–286.

[68] Keogh, E. and Kasetty, S. (2003). On the need for time series data mining benchmarks: a survey and empirical demonstration. *Data Mining and knowledge discovery*, 7(4):349–371.

[69] Kieu, T., Yang, B., Guo, C., and Jensen, C. S. (2019). Outlier detection for time series with recurrent autoencoder ensembles. In *IJCAI*, pages 2725–2732.

[70] Lacasa, L., Luque, B., Ballesteros, F., Luque, J., and Nuno, J. C. (2008). From time series to complex networks: The visibility graph. *Proceedings of the National Academy of Sciences*, 105(13):4972–4975.

[71] Lai, K.-H., Zha, D., Xu, J., Zhao, Y., Wang, G., and Hu, X. (2021). Revisiting time series outlier detection: Definitions and benchmarks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

[72] Lara-Benitez, P., Carranza-García, M., Garcia-Gutierrez, J., and Riquelme, J. C. (2020). Asynchronous dual-pipeline deep learning framework for online data stream classification. *Integrated Computer-Aided Engineering*, 27(2):101–119.

[73] Lee, S., Kwon, D., and Lee, S. (2003). Dimensionality reduction for indexing time series based on the minimum distance. *Journal of Information Science and Engineering*, 19(4):697–711.

[74] Levchenko, O., Kolev, B., Yagoubi, D.-E., Akbarinia, R., Masseglia, F., Palpanas, T., Shasha, D., and Valduriez, P. (2021). Bestneighbor: efficient evaluation of knn queries on large time series databases. *Knowledge and Information Systems*, 63(2):349–378.

[75] Li, H. (2019). Multivariate time series clustering based on common principal component analysis. *Neurocomputing*, 349:239–247.

[76] Li, H., Lin, C., Wan, X., and Li, Z. (2019). Feature representation and similarity measure based on covariance sequence for multivariate time series. *IEEE Access*, 7:67018–67026.

[77] Li, M. (2010). Fractal time series—a tutorial review. *Mathematical Problems in Engineering*, 2010.

[78] Li, Z., Zhao, Y., Han, J., Su, Y., Jiao, R., Wen, X., and Pei, D. (2021). Multivariate time series anomaly detection and interpretation using hierarchical inter-metric and temporal embedding. In *KDD*, pages 3220–3230. ACM.

[79] Lin, J., Keogh, E., Lonardi, S., and Chiu, B. (2003). A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, pages 2–11.

[80] Lin, J., Khade, R., and Li, Y. (2012). Rotation-invariant similarity in time series using bag-of-patterns representation. *Journal of Intelligent Information Systems*, 39(2):287–315.

[81] Lipponen, J. A. and Tarvainen, M. P. (2019). A robust algorithm for heart rate variability time series artefact correction using novel beat classification. *Journal of medical engineering & technology*, 43(3):173–181.

[82] Lu, Y., Cohen, I., Zhou, X. S., and Tian, Q. (2007). Feature selection using principal feature analysis. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 301–304.

[83] Magdy, N., Sakr, M. A., Mostafa, T., and El-Bahnasy, K. (2015). Review on trajectory similarity measures. In *2015 IEEE seventh international conference on Intelligent Computing and Information Systems (ICICIS)*, pages 613–619. IEEE.

[84] Markou, M. and Singh, S. (2003). Novelty detection: a review—part 1: statistical approaches. *Signal processing*, 83(12):2481–2497.

[85] Marussy, K. and Buza, K. (2013). Success: a new approach for semi-supervised classification of time-series. In *International Conference on Artificial Intelligence and Soft Computing*.

[86] McInnes, L. and Healy, J. (2017). Accelerated hierarchical density based clustering. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 33–42. IEEE.

[87] Megalooikonomou, V., Wang, Q., Li, G., and Faloutsos, C. (2005). A multiresolution symbolic representation of time series. In *21st International Conference on Data Engineering (ICDE'05)*, pages 668–679. IEEE.

[88] Melek, W. W., Lu, Z., Kapps, A., and Fraser, W. D. (2005). Comparison of trend detection algorithms in the analysis of physiological time-series data. *IEEE transactions on biomedical engineering*, 52(4):639–651.

[89] Morley, J. and Piger, J. (2012). The asymmetric business cycle. *Review of Economics and Statistics*, 94(1):208–221.

[90] Müller, M. (2007). Dynamic time warping. *Information retrieval for music and motion*, pages 69–84.

[91] Munir, M., Siddiqui, S. A., Dengel, A., and Ahmed, S. (2018). Deepant: A deep learning approach for unsupervised anomaly detection in time series. *Ieee Access*, 7:1991–2005.

[92] Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., and Yu, B. (2019). Definitions, methods, and applications in interpretable machine learning. *Proceedings of the National Academy of Sciences*, 116(44):22071–22080.

[93] Nanni, L., Brahnam, S., Ghidoni, S., Menegatti, E., and Barrier, T. (2013). Different approaches for extracting information from the co-occurrence matrix. *PloS one*, 8(12):e83554.

[94] Nanopoulos, A., Alcock, R., and Manolopoulos, Y. (2001). Feature-based classification of time-series data. *International Journal of Computer Research*, 10(3):49–61.

[95] Newman, M. (2010). *Networks: An Introduction*. Oxford University Press, Oxford.

[96] Nguyen, H., Tran, K. P., Thomassey, S., and Hamad, M. (2021). Forecasting and anomaly detection approaches using lstm and lstm autoencoder techniques with the applications in supply chain management. *International Journal of Information Management*, 57:102282.

[97] Oehmcke, S., Zielinski, O., and Kramer, O. (2015). Event detection in marine time series data. In *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*, pages 279–286. Springer.

[98] Paparrizos, J. and Gravano, L. (2015). k-shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 1855–1870.

[99] Paparrizos, J. and Gravano, L. (2016). K-shape: Efficient and accurate clustering of time series. *SIGMOD Record.*

[100] Park, H.-S. and Jun, C.-H. (2009). A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2):3336–3341.

[101] Pasupathi, S., Shanmuganathan, V., Madasamy, K., Yesudhas, H. R., and Kim, M. (2021). Trend analysis using agglomerative hierarchical clustering approach for time series big data. *The Journal of Supercomputing*, 77(7):6505–6524.

[102] Pena, D. and Box, G. E. (1987). Identifying a simplifying structure in time series. *Journal of the American statistical Association*, 82(399):836–843.

[103] Pena, E. H., de Assis, M. V., and Proença, M. L. (2013). Anomaly detection using forecasting methods arima and hwds. In *2013 32nd international conference of the chilean computer science society (sccc)*, pages 63–66. IEEE.

[104] Pincus, S. M. (1991). Approximate entropy as a measure of system complexity. *Proceedings of the National Academy of Sciences*, 88(6):2297–2301.

[105] Pliakos, K., Joo, S.-H., Park, J. Y., Cornillie, F., Vens, C., and Van den Noortgate, W. (2019). Integrating machine learning into item response theory for addressing the cold start problem in adaptive learning systems. *Computers & Education*, 137:91–103.

[106] Pokrajac, D., Lazarevic, A., and Latecki, L. J. (2007). Incremental local outlier detection for data streams. In *2007 IEEE symposium on computational intelligence and data mining*, pages 504–515. IEEE.

[107] Polikar, R. (2006). Ensemble based systems in decision making. *IEEE Circuits and systems magazine*, 6(3):21–45.

[108] Qin, Y., Ding, S., Wang, L., and Wang, Y. (2019). Research progress on semi-supervised clustering. *Cognitive Computation*, 11(5):599–612.

[109] Rakthanmanon, T., Campana, B., Mueen, A., Batista, G., Westover, B., Zhu, Q., Zakaria, J., and Keogh, E. (2012). Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 262–270.

[110] Rasamoelina, A. D., Adjailia, F., and Sinčák, P. (2020). A review of activation function for artificial neural network. In *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, pages 281–286. IEEE.

[111] Räsänen, T. and Kolehmainen, M. (2009). Feature-based clustering for electricity use time series data. In *International conference on adaptive and natural computing algorithms*, pages 401–412. Springer.

[112] Richman, J. S. and Moorman, J. R. (2000). Physiological time-series analysis using approximate entropy and sample entropy. *American Journal of Physiology-Heart and Circulatory Physiology*.

[113] Romano, S., Vinh, N. X., Bailey, J., and Verspoor, K. (2016). Adjusting for chance clustering comparison measures. *The Journal of Machine Learning Research*, 17(1):4635–4666.

[114] Rudin, C. (2019). Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215.

[115] Rätz, M., Javadi, A. P., Baranski, M., Finkbeiner, K., and Müller, D. (2019). Automated data-driven modeling of building energy systems via machine learning algorithms. *Energy and Buildings*, 202:109384.

[116] Sala, D. A., Jalalvand, A., Van Yperen-De Deyne, A., and Mannens, E. (2018). Multivariate time series for data-driven endpoint prediction in the basic oxygen furnace. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1419–1426.

[117] Salgado, C. M., Azevedo, C., Proença, H., and Vieira, S. M. (2016). Noise versus outliers. *Secondary analysis of electronic health records*, pages 163–183.

[118] Salvador, S. and Chan, P. (2007). Toward accurate dynamic time warping in linear time and space. *Intelligent Data Analysis*, 11(5):561–580.

[119] Samek, W., Montavon, G., Vedaldi, A., Hansen, L. K., and Müller, K.-R. (2019). *Explainable AI: interpreting, explaining and visualizing deep learning*, volume 11700. Springer Nature.

[120] Schäfer, P. (2015). The boss is concerned with time series classification in the presence of noise. *Data Mining and Knowledge Discovery*, 29(6):1505–1530.

[121] Schäfer, P. and Högqvist, M. (2012). Sfa: a symbolic fourier approximation and index for similarity search in high dimensional datasets. In *Proceedings of the 15th international conference on extending database technology*, pages 516–527.

[122] Schein, A. I., Popescul, A., Ungar, L. H., and Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260.

[123] Shatkay, H. and Zdonik, S. B. (1996). Approximate queries and representations for large data sequences. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 536–545. IEEE.

[124] Shih, S., Sun, F., and Lee, H. (2019). Temporal pattern attention for multivariate time series forecasting. *Mach. Learn.*, 108(8-9):1421–1441.

[125] Shmueli, G., Jank, W., Aris, A., Plaisant, C., and Shneiderman, B. (2006). Exploring auction databases through interactive visualization. *Decision Support Systems*, 42(3):1521–1538.

[126] Shokouhi, M. (2011). Detecting seasonal queries by time-series analysis. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 1171–1172.

[127] Solorio-Fernández, S., Carrasco-Ochoa, J. A., and Martínez-Trinidad, J. F. (2016). A new hybrid filter–wrapper feature selection method for clustering based on ranking. *Neurocomputing*, 214:866–880.

[128] Stahle, L. and Wold, S. (1989). Analysis of variance (anova). *Chemometrics and Intelligent Laboratory Systems*, 6(4):259–272.

[129] Steinley, D. (2004). Properties of the hubert-arable adjusted rand index. *Psychological methods*, 9(3):386.

[Thakkar et al.] Thakkar, P., Vala, J., and Prajapati, V. Survey on outlier detection in data stream. *Int. J. Comput. Appl*.

[131] Tiano, D., Bonifati, A., and Ng, R. (2021a). Featts: Feature-based time series clustering. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2784–2788.

[132] Tiano, D., Bonifati, A., and Ng, R. (2021b). Feature-driven time series clustering.

[133] Timmer, J., Gantert, C., Deuschl, G., and Honerkamp, J. (1993). Characteristics of hand tremor time series. *Biological cybernetics*, 70(1):75–80.

[134] Torres, J. F., Hadjout, D., Sebaa, A., Martínez-Álvarez, F., and Troncoso, A. (2021). Deep learning for time series forecasting: a survey. *Big Data*, 9(1):3–21.

[135] Van, G. A., Staals, F., Löffler, M., Dykes, J., and Speckmann, B. (2016). Multi-granular trend detection for time-series analysis. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):661–670.

[136] Van Craenendonck, T., Meert, W., Dumančić, S., and Blockeel, H. (2018). Cobras ts: A new approach to semi-supervised clustering of time series. In *International Conference on Discovery Science*, pages 179–193. Springer.

[137] Veltkamp, R. C. (2001). Shape matching: Similarity measures and algorithms. In *Proceedings International Conference on Shape Modeling and Applications*, pages 188–197. IEEE.

[138] Veltkamp, R. C. and Latecki, L. J. (2006). Properties and performance of shape similarity measures. In *Data Science and Classification*, pages 47–56. Springer.

[139] Ventura, D., Casado-Mansilla, D., López-de Armentia, J., Garaizar, P., López-de Ipina, D., and Catania, V. (2014). Ariima: a real iot implementation of a machine-learning architecture for reducing energy consumption. In *International conference on ubiquitous computing and ambient intelligence*, pages 444–451. Springer.

[140] Wang, H., Zhang, Q., Wu, J., Pan, S., and Chen, Y. (2019). Time series feature learning with labeled and unlabeled data. *Pattern Recognition*.

[141] Wang, X., Mueen, A., Ding, H., Trajcevski, G., Scheuermann, P., and Keogh, E. (2013a). Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309.

[142] Wang, X., Smith, K., and Hyndman, R. (2006). Characteristic-based clustering for time series data. *Data mining and knowledge Discovery*, 13(3):335–364.

[143] Wang, Y., Wang, P., Pei, J., Wang, W., and Huang, S. (2013b). A data-adaptive and dynamic segmentation index for whole matching on time series. *Proceedings of the VLDB Endowment*, 6(10):793–804.

[144] Wang, Z., Yan, W., and Oates, T. (2017). Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE.

[145] Wei, W. W. (2006). Time series analysis. In *The Oxford Handbook of Quantitative Methods in Psychology: Vol. 2*.

[146] Wipf, D. and Nagarajan, S. (2007). A new view of automatic relevance determination. *Advances in neural information processing systems*, 20.

[147] Xing, Z., Pei, J., Yu, P. S., and Wang, K. (2011). Extracting interpretable features for early classification on time series. In *Proceedings of the 2011 SIAM international conference on data mining*, pages 247–258. SIAM.

[148] Ye, L. and Keogh, E. (2009). Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 947–956.

[149] Zhang, W., Dong, X., Li, H., Xu, J., and Wang, D. (2020). Unsupervised detection of abnormal electricity consumption behavior based on feature engineering. *IEEE Access*, 8:55483–55500.

[150] Zhao, B., Lu, H., Chen, S., Liu, J., and Wu, D. (2017). Convolutional neural networks for time series classification. *Journal of Systems Engineering and Electronics*, 28(1):162–169.

[151] Zhu, X. and Goldberg, A. B. (2009). Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130.

[152] Zoumpatianos, K., Idreos, S., and Palpanas, T. (2016). Ads: the adaptive data series index. *The VLDB Journal*, 25(6):843–866.