POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Anomalearn: a modular and extensible library for the development of time series anomaly detection models

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Marco Petri**

Student ID: 963502
Advisor: Prof. Piero Fraternali
Co-advisors: Nicolò Oreste Pinciroli Vago
Academic Year: 2021-2022

Abstract

Anomaly detection is the problem of identifying abnormally and potentially dangerous or faulty behaviour. It is applied in various domains and on several data types: tabular, images, or temporal. Many current state-of-the-art anomaly detection algorithms for time series share preprocessing or postprocessing operations. Furthermore, several datasets are employed for evaluating and comparing the performance of models. Such benchmark datasets exhibit a variable degree of complexity, which may affect the evaluation of the power of the methods compared in distinct experiments. Some publicly available datasets are overly simple and their use may overestimate the power of the models tested on them. However, there is no automatic and unambiguous definition of simplicity for these datasets. The first contribution of this thesis assesses the problem of simplicity in datasets of time series anomaly detection through a formal approach. It reports a definition of simplicity for anomaly detection datasets and the definition of scores representing three different types of simplicity. It also proposes algorithms to compute these scores on datasets and the analysis of their time complexity. Secondly, although libraries have been presented for many tasks, there is still a lack of an extensible and modular library specifically developed for creating new techniques related to anomaly detection. This thesis proposes anomalearn, a library for developing new models and approaches for time series anomaly detection. It consists of a rigorous object-oriented design using UML as the first tool to describe its functioning. Furthermore, anomalearn uses an approach based on interfaces to share API design employing the Python programming language, the current standard for machine and deep learning solutions. The thesis source code can be downloaded and explored at: https://github.com/marcopetri98/2021-2022-thesis.

Keywords: anomaly detection, time series, datasets, library, object oriented programming, python

Abstract in lingua italiana

Il rilevamento delle anomalie è il problema riguardante l'identificazione di comportamenti abonormi che possono rappresentare un danno o un guasto. La sua applicazione avviene in svariati domini di applicazione e mediante diversi tipi di dato: tabulari, temporali o di immagini. Molti degli algoritmi stato dell'arte di rilevamento delle anomalie per serie temporali hanno in comune le fasi di pre-processamento e post-processamento. Inoltre, diversi insiemi di dati vengono utilizzati da numerosi approcci per valutare e confrontare le prestazioni dei modelli. Infine, alcuni insiemi di dati pubblici sono semplici e non c'è una definizione evidente e automatica di semplicità per questi insiemi. La prima contribuzione di questa tesi è valutare il problema della semplicità in questi insiemi per il rilevamento delle anomalie nelle serie temporali mediante un approccio formale. Essa riporta una definizione formale di semplicità per insiemi di dati per il rilevamento delle anomalie e la definizione di punteggi rappresentanti tre diversi tipi di semplicità. Inoltre, essa propone algoritmi per calcolare i suddetti punteggi su insiemi di dati e l'analisi della loro complessità temporale. Secondariamente, anche se delle librerie sono state presentate per svolgere svariati problemi, c'è ancora una carenza sostanziale di una libreria estendibile e modulare specificamente sviluppata per la creazione di nuove tecniche di rilevamento delle anomalie. Questa tesi propone anomalearn, una libreria per lo sviluppo di nuovi modelli e approcci per il rilevamento di anomalie nelle serie temporali. Essa consiste in un progetto rigoroso orientato agli oggetti utilizzante UML come primo strumento di descrizione delle sue funzionalità. In aggiunta, anomalearn utilizza un approccio basato su interfacce per condividere il disegno delle API e il linguaggio di programmazione Python, lo standard de facto per soluzioni di apprendimento automatico e profondo. Il codice sorgente della tesi può essere scaricato ed esplorato all'indirizzo: https://github.com/marcopetri98/2021-2022-thesis.

Parole chiave: rilevamento delle anomalie, serie temporali, insiemi di dati, libreria, programmazione orientata agli oggetti, python

Contents

Acronyms

Acc Accuracy. 34, 35, 42

AD Anomaly Detection. 2, 5, 26–30, 33–37, 39, 41, 42, 46, 49, 51, 52, 58, 61, 62, 64, 69–72, 75, 99, 105

AIRAI Add If Removable And Independent. 67, 68

AUC Area Under the Curve. 36

AUC-PR Area Under the Presicion-Recall Cruve. 36, 37

AUC-ROC Area Under the Receiver Operating Characteristics Curve. 36, 37

BAcc Balanced Accuracy. 35

CCE Categorical Cross Entropy. 34, 35

CDF Cumulative Density Function. 6, 93, 107

DRY Don't Repeat Yourself. 58

F$_\beta$ F$_\beta$ Score. 36–38

FPR False Positive Rate. 35, 37

MSE Mean Squared Error. 33

PDF Probability Density Function. 5–7, 93, 94, 107

PMF Probability Mass Function. 93

Pre Precision. 34, 36, 37

Rec Recall. 34, 36, 37

SCCE Sparse Categorical Cross Entropy. 34

TAGRI They Ain't Gonna Read It. 55

TNR True Negative Rate. 34, 45

TPR True Positive Rate. 34, 35, 37, 45

WCCE Weigthed Categorical Cross Entropy. 35

1 | Introduction

The work of this thesis is about time series anomaly detection, which means that the emphasis is on two items: the type of data (time series) and the task carried out on this data (anomaly detection). The kind of data is important in the context of industrial (and other fields) measurements, which are related to some ordering of time; between all the application fields of time series, some standard and notable examples are weather forecasting and stock market prediction. Time series analysis is quite an old topic with a massive amount of published literature about it, of which some is the theory of the generation process of a time series through the theoretical element called stochastic processes (see chapter 2). The historical fields of observation of time series were fields of study in which the collection of measurements was extremely serviceable: weather, economics, medicine and astronomy [62]. The great interest and growth in time series datasets can be observed in many ways, one of which is browsing the UCI repository [27] (storage of datasets) containing 131 time series datasets for different purposes. More recently, datasets containing time series about industrial equipment to carry on diverse tasks (like predictive maintenance and anomaly detection) have been published. Time series are employed in several problems: forecasting, classification, clustering, anomaly detection, and many others. In this thesis, anomaly detection for time series is studied and used to propose metrics to evaluate the complexity of datasets. Secondly, a library for developing anomaly detection solutions for time series is proposed. The task of anomaly detection is characterized by the presence of two types of data: normal and anomalous. The anomalous data may represent several behaviours and is a minuscule portion with respect to the amount of normal data. The anomalies may depict faulty behaviour in industrial machines, cyber attacks in systems, or possible signs of illness in patients through the analysis of MRIs or ECGs. Therefore, the ability of an anomaly detection method to accurately identify anomalies and raise alarms/messages is of great importance and interest, e.g., the identification of signs for tumours in MRIs is crucial to start treating the patients with appropriate cures as soon as it is possible.

The current state-of-the-art in time series greatly varies depending on the task (classification, forecasting or others). Overall, there is a shift from statistical and machine

learning solutions to deep learning approaches. Neural networks are increasingly popular for developing solutions for time series in many fields. Particularly, with the introduction of transformers [79], new powerful neural network blocks capable of extracting strong temporal correlations are available. Regarding anomaly detection specifically, there are few libraries encompassing the problem of containing several state-of-the-art models to aid data scientists in comparing existing and newly proposed approaches. There are libraries for deep learning solutions for anomaly detection [6], for tabular data solutions for anomaly detection [89], and for time series unsupervised anomaly detection [7]. However, none of the published libraries proposes a methodological, clear, and simple way to develop new methods providing UML diagrams and well-structured interfaces by using the software engineering best practices. In general, in the context of machine and deep learning solutions, there are libraries for developing solutions with well-detailed documentation for each component, but none of these libraries is meant to create general solutions for anomaly detection. Moreover, aside from the lack of libraries for developing anomaly detection solutions, there is also the problem of datasets being simple and not providing a good metric for evaluating the efficacy of techniques.

This thesis aims to fill some of the gaps in the current state-of-the-art libraries for anomaly detection for time series and to provide definitions and algorithms for evaluating the simplicity of published datasets for anomaly detection. The proposed library aims at aiding the data scientist in the creation of new approaches, from data processing to models for time series anomaly detection. The simplicity definitions and algorithms aim at supplying the scientific community with tools to evaluate whether a dataset is or is not simple according to some statistical measures.

The work on the evaluation of the complexity of an AD dataset is inspired by the work in [85]. The work in [85] provides an insightful view of the inherent problem of simple datasets available to the scientific community, and proposes a method to evaluate whether a dataset is trivial. This thesis applies a similar analysis to the complexity of a dataset relative to the specific AD task by proposing a definition of simplicity in terms of spatial properties. On top of that definition, scores for evaluating the degree of simplicity of datasets are presented beside the algorithms for their computation.

Anomalearn (the proposed library) enables the data scientist to develop anomaly detection models and processing methods for time series in an easy way. The data scientist can decide whether to use single modules separately or together to reduce the number of functionalities to implement, especially the frequent ones. It also ships functionalities to aid the creation of experiments on datasets using defined data readers to automatically create a chain of experiments with automatic train/test splits, such that the data scientist

can effortlessly state the datasets along with their location on disk and splitting, and use these declarations to iterate over the datasets to perform experiments in few lines of code. Furthermore, algorithms to evaluate the simplicity of datasets are shipped with the library, which enables the data scientist to evaluate the simplicity of datasets; it enables the selection of simple and complex time series to assess a model through experiments.

Chapter 2 introduces all the necessary concepts to understand the rest of the thesis. Chapter 3 introduces the study of simplicity with definitions and algorithms for computation, along with the determination of the degree of simplicity of publicly available datasets. Chapter 4 introduces the proposed library for developing solutions for time series anomaly detection. It also compares the proposed library to existing solutions in anomaly detection and general-purpose machine learning libraries. Chapter 5 draws the conclusions of this thesis work, while chapter 6 lists some of the possible tracks for further research. Finally, appendix A introduces necessary definitions of topics related to statistics and probability to understand topics in chapter 2, and appendix B introduces the notation to identify code elements in written text.

2 | Background and state-of-the-art

This section introduces all the concepts needed to read this thesis in a brief summary and a review of the state-of-the-art methods. Only the notions that are directly used by state-of-the-art methods or that are directly used by thesis specific work are introduced in this chapter. In case the reader does not know some of the chapter's underlying topics, the appendix A contains more detailed and basic concepts.

Throughout this chapter, several sources will be used as reference. Moreover, since some sections continuously use the same sources, they are listed here to avoid continuous repetition of the same citation:

- Statistics and probability: [24, 38, 66] are the main sources for this section.
- Time series: [13, 43, 53, 61] are the main sources for this section.

2.1. Statistics and probability

In the context of Anomaly Detection (AD), it is frequent that a method decides whether a point is anomalous based on a threshold. Several approaches can compute such a decision boundary, the work in [59] presents a multivariate gaussian distribution to model errors and calculate a threshold over the gaussian PDF (section 2.6 contains a detailed explanation). This chapter briefly introduces the Gaussian distribution aiming at introducing its generalization to random vectors (i.e., a multivariate gaussian distribution).

> **Definition 2.1.1** (Gaussian distribution)**:** *Let X be a continuous random variable, $\mu \in \mathbb{R}$, and $\sigma^2 \in \mathbb{R}_+$. We say that X has a gaussian distribution if its*

Probability Density Function is:

$$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\} \tag{2.1}$$

We denote this distribution by $X \sim \mathcal{N}(\mu, \sigma^2)$

The Gaussian distribution is also called the Normal distribution. One of its interesting properties is that any distribution $\mathcal{N}(\mu, \sigma^2)$ is transformable in $\mathcal{N}(0, 1)$. It follows that being able to compute the probability of $\mathcal{N}(0, 1)$ enables the computation of the probability of a Gaussian random variable with any mean and variance.

Proposition 2.1.1: *Let* X *be a normal random variable and* $a, b \in \mathbb{R}$ *s.t.* $b \neq 0$. *Then, the random variable* $Y = a + bX$ *is a normal random variable* $Y \sim \mathcal{N}(a + b\mu, b^2\sigma^2)$.

It follows the following proposition:

Proposition 2.1.2: *Let* X *be a normal random variable* $X \sim \mathcal{N}(\mu, \sigma^2)$. *Then the random variable* $Z = (X - \mu)/\sigma$ *is a normal random variable* $Z \sim \mathcal{N}(0, 1)$

If a normal random variable has distribution $\mathcal{N}(0, 1)$, it is said to have a standard normal distribution.

While dealing with normal random variables, it is common to use a process called *standardization*: the transformation defined by proposition 2.1.2. The standard normal distribution is so important that its CDF is identified by $\Phi(x)$, and its PDF is identified by $\phi(x)$.

Theorem 2.1.1 (Central limit theorem)**:** *Let* $X_1, X_2, ...$ *be any sequence of independent identically distributed random variables with mean* μ *and finite positive variance* σ^2, *and let* $\overline{X}_n$ *be their average. Then for any* $x \in \mathbb{R}$*:*

$$\lim_{n \to +\infty} P\left(\frac{\sqrt{n}(\overline{X}_n - \mu)}{\sigma} \leq x\right) = \Phi(x) \tag{2.2}$$

where Φ *is the CDF of* $\mathcal{N}(0, 1)$.

The *central limit theorem* is an significant result in statistics, and it ahs to do with normal

distribution. Its interpretation is that given a sufficiently large number of independent identically distributed random variables, the standardization of their average (or sum) is approximately distributed as a standard normal variable. It is the justification with which many research works can assume that the distribution of noise or errors is normal.

Given these few meaningful properties, it is possible to introduce the multivariate gaussian distribution, i.e., gaussian distribution over random vectors instead of random variables.

> **Definition 2.1.2** (Multivariate gaussian distribution)**:** *Let $X \in \mathbb{R}^n$ be an n-dimensional random vector. X is a gaussian random vector if and only if, $\forall a \in \mathbb{R}^n$ the random variable $a^T X$ is gaussian. The notation $\mathcal{N}(\mu, \Lambda)$ is used to denote that X has (multivariate) gaussian distribution with mean vector μ and covariance matrix Λ.*

By definition, the covariance matrix may be singular; in such a case, the PDF is not defined. However, only multivariate gaussian distributions whose matrix is not singular are of interest. Each component of a Gaussian random vector is a Gaussian random variable, their sum is a Gaussian random variable, and the marginal distributions are Gaussian. Furthermore, since the components of the vector weren't assumed independent, they may even be dependent on each other.

The deep treatment of the estimation of a multivariate gaussian distribution (mean vector and covariance matrix) is out of the scope of this section.

2.2. Time series

Time series are the central component of this thesis. Time series can be used to operate anomaly detection on FRIDGE (a private dataset available to this thesis work) and other publicly available datasets. When possible, FRIDGE is used to test components of the presented library (see chapter 4). This section introduces univariate time series, multivariate time series, and the differences between these two types of time series.

The fundamental piece defining time series is a stochastic process. A stochastic process is a sequence of random variables ordered with respect to an index t, often referred to as time. A stochastic process is a sequence of random variables defined on the same sample space. Formally:

Definition 2.2.1 (Stochastic process [13, 53, 61])**:** *Let $\mathcal{T} \subseteq \mathbb{R}$ be a set, and let $(S, \mathfrak{F}, P)$ be a probability space. A stochastic process is a set of random variables over $(S, \mathfrak{F}, P)$ defined as:*

$$V = \{v(t, s) | t \in \mathcal{T} \wedge s \in S\} \tag{2.3}$$

Where $v : \mathcal{T} \times S \to \mathbb{R}$ is a function.

In other words, a stochastic process is a function of the index and the sample space. By fixing $t = t_0$, the stochastic process becomes a function solely of the event $v(s)$. Conversely, fixing the event $s = s_0$ converts it into a function solely of the index $v(t)$ called *process realization*. The reader might note that a time series is the process realization of a stochastic process. Recall that $\mathfrak{F}$ is a σ-algebra [9, 38].

It is frequent that models work on a specific type of stochastic process: stationary stochastic processes. The definition of stationarity for a stochastic process requires the introduction of some properties.

Definition 2.2.2 (Mean function [13, 53, 61])**:** *Let V be a stochastic process defined over the index set $\mathcal{T} \subseteq \mathbb{R}$ and probability space $(S, \mathfrak{F}, P)$. The function $m : \mathcal{T} \to \mathbb{R}$ is called mean function, and it is defined as:*

$$m(t) = E[v(t, \cdot)] \tag{2.4}$$

Definition 2.2.3 (Variance function [13])**:** *Let V be a stochastic process defined over the index set $\mathcal{T} \subseteq \mathbb{R}$ and probability space $(S, \mathfrak{F}, P)$. The function $Var : \mathcal{T} \to \mathbb{R}$ is called variance function, and it is defined as:*

$$Var(t) = E\left[(v(t, \cdot) - m(t))^2\right] \tag{2.5}$$

Definition 2.2.4 (Covariance function [13, 53, 61])**:** *Let V be a stochastic process defined over the index set $\mathcal{T} \subseteq \mathbb{R}$ and probability space $(S, \mathfrak{F}, P)$. The function $\gamma : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$ is called covariance function, and it is defined as:*

$$\gamma(t_1, t_2) = E\left[(v(t_1, \cdot) - m(t_1))\,(v(t_2, \cdot) - m(t_2))\right] \tag{2.6}$$

These definitions are the adaptation of mean, variance and covariance functions of random

variables for stochastic processes. Recall that by fixing the time for a stochastic process, a random variable is obtained. However, from now on, I will use the notation $v(t)$ to refer to the random variable $v(t,\cdot)$ of a stochastic process for the sake of simplicity. Now, stationarity can be defined:

Definition 2.2.5 (Strict stationarity [53])**:** *Let V be a stochastic process defined over the index set $\mathcal{T} \subseteq \mathbb{R}$ and probability space $(S, \mathfrak{F}, P)$. If for every $t_1, ..., t_n \in \mathcal{T}^n$ and $\tau \in \mathbb{R}$ the distribution of the random vector $[v(t_1+\tau), ..., v(t_n+\tau)]$ is independent of τ, the process is strictly stationary.*

Definition 2.2.6 (Weak stationarity [53])**:** *Let V be a stochastic process defined over the index set $\mathcal{T} \subseteq \mathbb{R}$ and probability space $(S, \mathfrak{F}, P)$. If the mean function $m(t)$ is constant and its covariance function $\gamma(t_1, t_2)$ depends upon $t_2 - t_1$ only, then the process is weakly stationary.*

The definition of a weakly stationary process implies that its variance function is constant. It is a direct consequence of the definition of weak stationarity because the covariance function depends only on the time lag; in fact, it can be written as:

$$\gamma(\tau) = E\left[(v(t) - m))\,(v(t+\tau) - m)\right] \tag{2.7}$$

From now on, the weak stationarity assumption will be used many times for stochastic processes. Furthermore, before introducing some relevant properties of a weakly stationary process, the correlation coefficient for a stochastic process must be defined.

Definition 2.2.7 (Correlation function [13])**:** *Let V be a stochastic process defined over the index set $\mathcal{T} \subseteq \mathbb{R}$ and probability space $(S, \mathfrak{F}, P)$, and let γ be its covariance function. The function $\rho : \mathcal{T} \times \mathcal{T} \to \mathbb{R}$ is called correlation function, and it is defined as:*

$$\rho(t_1, t_2) = \frac{\gamma(t_1, t_2)}{\sqrt{\gamma(t_1, t_1)}\sqrt{\gamma(t_2, t_2)}} \tag{2.8}$$

Similarly to the covariance function, the correlation function of a weakly stationary process depends upon the lag only since it is defined utilizing the covariance function. It means

that for a weakly stationary process, it is:

$$\rho(\tau) = \frac{\gamma(\tau)}{\sqrt{\gamma(0)}\sqrt{\gamma(0)}} = \frac{\gamma(\tau)}{\gamma(0)} \tag{2.9}$$

Some [13] call the correlation function of a weakly stationary process *normalized covariance function*, and so I will. Note that the correlation function is not defined if $\gamma(0) = 0$, and it does not affect its usage. If the covariance function is zero, data are equal to each other and constant, which is useless and impossible in real-world scenarios since there is at least some noise. The covariance function of a weakly stationary process has three paramount properties:

Proposition 2.2.1: *Let γ be the covariance function of a weakly stationary process. The covariance function has the following properties:*

- $\gamma(0) \geq 0$
- $\gamma(-\tau) = \gamma(\tau)$
- $|\gamma(\tau)| \leq \gamma(0)$

Now, the final background component can be defined. It is a cardinal type of process commonly used to describe the noise. The reason for its importance is that it is unpredictable.

Definition 2.2.8 (White process [13])**:** *Let η be a weakly stationary process. If its covariance function is $\gamma(\tau) = 0$ for every $\tau \neq 0$, we call it white process and we write $\eta \sim WN(\mu, \sigma^2)$ stating that μ is its mean and σ^2 is its variance.*

The white process is the essential element used for other processes' definitions. Measurement experiments discovered that the white process is fruitful in modelling noise in data.

2.3. Time series decomposition

Given the definitions and properties related to stochastic processes, it is possible to investigate the properties of time series. Recall that a time series is the process' realization of a stochastic process, i.e., the result of an experiment performed over a stochastic process. Since a process' realization is a function of time, time series may exhibit two behaviours: trend and periodicity. If a time series has trend and periodicity, it can't be assumed to be stationary, and since stationarity is a common assumption in many methods, it is nec-

essary to scrutinize the relationship between trend, periodicity and stationarity. Hence, the components of a time series are described and divided into trend, seasonal and cyclic.

Definition 2.3.1 (Trend [43])**:** *Let Y be a time series. We call trend an increasing or decreasing component in the time series.*

Definition 2.3.2 (Seasonality [43])**:** *Let Y be a time series. We call seasonality a periodic component (known period) in the time series.*

Definition 2.3.3 (Cyclic [43])**:** *Let Y be a time series. We call cyclic a component that show rises or falls in the time series without exact frequency.*

The periodic component is called seasonality because many application fields of time series have patterns influenced by the calendar (e.g., the hour of the day, the day of the week, the month, the season and so on). However, since a seasonality is a periodic component, it may also contain periodicity independent of the calendar. A time series can have multiple seasonalities and multiple cyclic behaviour. Differently, the time series has only one trend component that may have change-points (i.e., the time series can either have a positive, null or negative trend in a time interval, it cannot rise and fall concurrently).

The time series decomposition in the basic blocks is useful in many ways, e.g., to obtain a stationary series given a nonstationary series. Even if each component can be found independently, many decomposition methods group the trend and the cycle together. Therefore, methods can decompose the time series into four components in an additive way:

$$y(t) = S(t) + T(t) + C(t) + R(t) = S(t) + TC(t) + R(t) \tag{2.10}$$

Where $S(t)$ is the seasonal component, $T(t)$ is the trend component, $C(t)$ is the cyclic component, and $R(t)$ is the residual component (what remains of the time series after we removed the others). However, many decomposition methods group the trend and cyclic components into the trend-cycle component $TC(t)$. Alternatively, another common way to decompose the time series is the multiplicative method:

$$y(t) = S(t) \cdot T(t) \cdot C(t) \cdot R(t) = S(t) \cdot TC(t) \cdot R(t) \tag{2.11}$$

The multiplicative method is preferable when the seasonal and trend-cycle components appear proportional to the level of the time series [43], i.e., the average value in an interval, the component captured by the trend-cycle. Conversely, when the components are not,

the additive approach is preferred.

2.3.1. Decomposition methods

There exist several methods to decompose a time series. The simplest decomposition method is *moving average* decomposition. It is a method to smooth the original time series using the average over a moving window. Besides the algorithm description, it is also essential to clearly define what a moving average is:

Definition 2.3.4 (Moving Average)**:** *Let $y(t)$ be a time series. We write m-MA to identify the moving average of order m of the time series $y(t)$, and it is defined as:*

$$\hat{T}(t) = \begin{cases} \frac{1}{m} \sum_{i=-\frac{m-1}{2}}^{\frac{m-1}{2}} y(t+i) & \text{if } m = 2k+1 \wedge m \geq 1 \wedge k \in \mathbb{N} \\ \frac{1}{m} \sum_{i=-\frac{m}{2}}^{\frac{m}{2}-1} y(t+i) & \text{if } m = 2k \wedge \frac{m}{2} \geq 1 \wedge k \in \mathbb{N} \setminus 0 \end{cases} \tag{2.12}$$

When the moving average is applied to a time series, it smooths the time series and reduces its dimension by $n-1$, where n is the order of the moving average. Generally, the order of the moving average order is odd. However, not only it is possible to use an even moving average, it is possible to perform an even moving average on an even moving average to obtain a centred even moving average. In the latter case, the process is written as $m_2 \times m_1$-MA, where m_2 is the order of the moving average to be applied to the first moving average. The decomposition of a time series using a moving average is composed of several steps:

1. Compute the trend-cycle component. If m is odd, use a m-MA for the estimation; otherwise, use a $2 \times m$-MA to compute a centered moving average.
2. Compute the detrended series. If the method is additive, it is $y(t) - TC(t)$; otherwise, it is $y(t)/TC(t)$.
3. Compute the seasonal component from the detrended component. Fix the length of the season to any value $n \geq 2$. If the season length is n, the i$^{\text{th}}$ element of the season is the average of every observation of the i$^{\text{th}}$ elements of non-overlapping sequences of length n.
4. Compute the remainder component. If the method is additive, it is $y(t) - TC(t) -$

$S(t)$; otherwise, it is $y(t)/TC(t)S(t)$.

This method does not represent the state-of-the-art of decomposition methods, and it has several problems [43]: some initial and starting observations cannot be decomposed (due to moving average), rapid rises and falls are over-smoothed, seasonality is assumed not to change in the time series, and the decomposition is not robust to anomalous data. Therefore, there has been an effort in the scientific community to develop more advanced decomposition methods.

2.3.2. STL Seasonal and Trend decomposition using Loess

The work in [22] introduces STL to decompose a time series into three components: trend-cycle, seasonal and residual. This method decomposes the time series using an additive approach. Therefore, obtaining a multiplicative decomposition requires transforming the data before using the decomposition method. Figure 2.1 provides an example of decomposition using a FRIDGE dataset.

Loess (locally-weighted regression) is a technique which can be used to smooth a curve. Given a set of independent (x) and dependent (y) variables, a local curve is fitted to the data at each location (x, y) to get the Loess value for that point. Loess has two parameters: the neighbour dimension $q \in \mathbb{N} \setminus \{0\}$ (the number of elements in the neighbourhood) and the degree of the polynomial to fit d. If the series to be smoothed has n points, the following quantity is defined (a function of the independent variable or variables):

$$\lambda_q(x) = \begin{cases} \text{distance of q}^{\text{th}} \text{ farthest point from x} & q \leq n \\ \lambda_n(x)\dfrac{q}{n} & q > n \end{cases} \tag{2.13}$$

This function defines the neighbourhood of a point to perform the local regression. However, the definition of the neighbourhood of a point also uses the tricube function, which is:

$$B(u) = \begin{cases} (1-u^3)^3 & 0 \leq u < 1 \\ 0 & u \geq 1 \end{cases} \tag{2.14}$$

Finally, if the data are ordered with respect to some index i, the weight of the i^{th} data point around x is defined as:

$$v_i(x) = B\left(\frac{|x_i - x|}{\lambda_q(x)}\right) \tag{2.15}$$

These weights are used to define the points to be used to perform the local regression. The tricube function enables locality since its argument will always be greater than zero and since it is a weight laying in $[0, 1]$. Therefore, the closer the point, the bigger the weight.

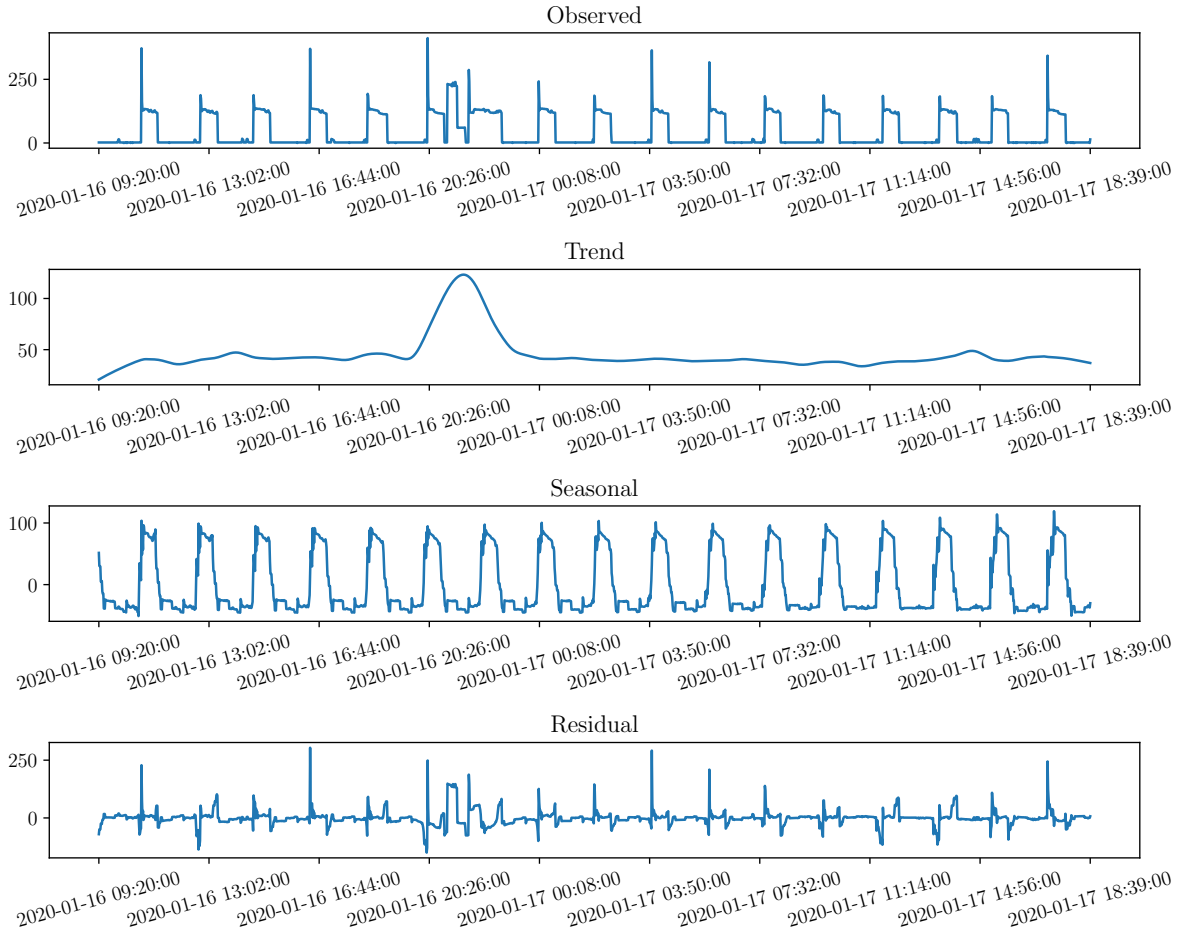


Figure 2.1: Example of time series decomposition using STL [22] on a FRIDGE. The image can be generated by running the script "example_stl.py".

As q goes to infinity, the polynomial to be fit converges to the least-squares estimate of a polynomial. Moreover, as q goes to infinity, the estimated polynomial is no more local since weights tend to 1 independently by the point around which the regression is computed.

The decomposition of the series into trend, seasonal and residual components is divided into two loops: an outer loop and an inner loop. The outer loop is used in case one wants to perform a robust estimation of the trend and seasonal components in case some outliers are present in the data. The inner loop performs the decomposition using of Loess smoothing and moving averages. Specifically:

- Inner loop: computes the seasonal component using Loess from the detrended series and applies a low-pass filter (made up of moving averages and Loess) to the seasonal component. Then, it computes the trend using Loess from the deseasonalized series.

- Outer loop: calculates the residual by removing the trend and the seasonal from the original time series

The computation of the seasonal component in the inner loop uses the detrended series and vice-versa (trend computation uses deseasonalized series). It is possible because the algorithm initializes the series to be all zeros at the first run. Then, it uses the detrended series to compute the seasonal component to allow robust decomposition.

2.3.3. New decomposition approaches

The research is still active on time series decomposition. Although STL is capable of decomposing a time series into trend-cycle, seasonal and residual components, it is also affected by some weaknesses: it works only with additive methods (needs transformation and back-transformation to use the multiplicative decomposition), it does not take into account exogenous variables, it does not consider multiple seasonalities at the same time, and it is sensible to abrupt changes in trend. The work in [25, 26] present STR and Robust STR aiming to solve the problem of the inability to estimate multiple seasonal components. Real-world time series have complex patterns mainly due to the presence of various overlapping phenomena, which may also be seasonal with different periodicities [25]. Separately capturing all these periodicities is an appealing task. Their model allows decomposing a time series with any number of seasonal components and exogenous variables with constant, time-varying or seasonal coefficients. Even though this method provides these improvements, it has a drawback: it is slow to compute. The work in [82] presents RobustSTL, an approach aiming at improving previous methods by allowing the modelling of abrupt change in trend, keeping an eye on anomaly detection. Because of this interest, they decompose a time series in trend-seasonal-residual while assuming the residual component to be composed of white noise, spikes, and dips. Their decomposition method is able to capture cleaner and smoother seasonal components while maintaining the point anomalies: it does not eliminate outliers. It allows keeping point anomalies since it uses bilateral filtering for noise removal [82] and the least absolute deviation loss for trend estimation. Therefore, it is more suitable for anomaly detection. However, this approach still does not allow the computation of multiple seasonalities. To enable the estimation of several seasonalities in RobustSTL, the work in [83] presents FastRobustSTL. It significantly improves the computational complexity of the algorithm from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$.

Besides introducing such approaches, it is worth mentioning that the decomposition methods implemented in Python libraries are STL and standard moving average decomposition,

which are implemented in statsmodels [68]. The other approaches do not have official (by the authors) implementation in python:

- STR [26]: is implemented in R and is distributed as an R package.
- RobustSTL [82]: is not officially implemented. However, although an unofficial implementation exists [51], it is not delivered as Python package.
- FastRobustSTL [83]: has no public implementation.

2.4. ARIMA processes

Decomposition and theory on time series are fundamental concepts. However, research also involves the actual computation of models for time series aiming to capture their generation process. Among all the existing models, SARIMAX models (a class of models) use a white process as a fundamental block to create a statistical model for a time series. Such models work for either univariate or multivariate time series in which there is interest only in predicting one feature. A statistical model treating multivariate time series exists, and it is called VARMAX; however, it is not the focus of this section for two reasons: in most real-world datasets, either we are interested in one channel prediction (e.g., in [41] only telemetry is of interest), or we have too many dimensions (e.g., in [44] there are thousands of features), and machine learning and deep learning outperforms statistical methods in many tasks.

The SARIMAX model family includes other families by themselves, one of which is the ARIMA model family (before introducing it formally, the *prediction problem* must be introduced). It is both a model for time series and a stochastic process since it can be proved to be identical to an ARMA model/process.

The prediction problem is the task of predicting future values based on past data. In its simplest form, it presents itself as the one-step-ahead prediction problem (predicting the value of the next point). In the context of time series data, this task is often called forecasting. Several domains rely on forecasting: stock price forecasting, weather forecasting, biomedical signal forecasting and many others. Let us call $v(t)$ a variable representing a process realization. The forecasting problem in its simplest form (one-step-ahead) is to predict the value $v(t+1)$, that is:

$$\hat{v}(t+1|t) = f(v(1), v(2), ..., v(t)) \tag{2.16}$$

Where $\hat{v}$ represent a prediction, v represent the real value of the function, and f is the

model used to predict future values based on present and past data. This function won't be exact and will yield errors in prediction. These errors can be computed by simply measuring the error between the real value and the predicted value for a given time instant:

$$\varepsilon(t+1) = v(t+1) - \hat{v}(t+1|t) \tag{2.17}$$

Theoretically, the best model yields $\varepsilon(t+1) = 0$ by perfectly predicting future values. However, achieving this result is practically impossible because of noise. The best obtainable predictors yield a fully unpredictable error: it is impossible to predict the error it will have on forecasted values. Namely, the best predictor has $\varepsilon \sim WN(\mu, \sigma^2)$. Note that if a model has predictable prediction error, it is possible to build another model predicting the error to obtain an overall model yielding either another model with a predictable prediction error or a model with an unpredictable prediction error.

2.4.1. MA process

A Moving Average process is a linear combination of current and past values of a white noise term. Those white noise terms can be interpreted as the prediction errors made by the model.

> **Definition 2.4.1** (Moving Average Process [13])**:** *Let $\eta \sim WN(0, \sigma^2)$ be a white noise, and let $c_0, c_1, ..., c_q \in \mathbb{R}$ be parameters. We call moving average process of order q, and we write $MA(q)$, a stochastic process defined as:*

$$y(t) = \sum_{i=0}^{q} c_i \eta(t-i) \tag{2.18}$$

There exist several different definitions of the moving average process. The previous is only one of the currently existing definitions. Other definitions of the moving average

process of order q are:

$$y(t) = \eta(t) + \sum_{i=1}^{q} c_i \eta(t-i) \tag{2.19}$$

$$y(t) = \eta(t) - \sum_{i=1}^{q} c_i \eta(t-i) \tag{2.20}$$

$$y(t) = \mu + \eta(t) + \sum_{i=1}^{q} c_i \eta(t-i) \tag{2.21}$$

All having $\eta \sim WN(0, \sigma^2)$. Let $y_1(t)$ be the first alternative definition, and $y_3(t)$ be the third. It can be immediately observed that $y_3(t) = y_1(t) + \mu$. It means they have the same variance and covariance functions. The only difference between the two definitions is the mean function. However, since the mean can be any real value (also 0) and a process can be defined on top of another by shifting the mean, they are identical in terms of theoretical properties.

Let $y_2(t)$ be the second optional definition. It has the exact same white noise terms and parameter's position and number with respect to $y_1(t)$. Since the constants can be any real number, the equivalence between the two is $c_{i,1} = -c_{i,2}$ where $c_{i,j}$ is the constant i of definition j. Therefore, they have the exact same properties.

Therefore, proving that all the definitions are the same can be reduced to proving $y_1(t) = y(t)$. Let $\xi(t) = c_0 \eta(t)$, the first definition of moving average process becomes:

$$y(t) = \xi(t) + \sum_{i=1}^{q} \frac{c_i}{c_0} \xi(t-i) = \xi(t) + \sum_{i=1}^{q} d_i \xi(t-i) \tag{2.22}$$

Where $\xi \sim WN(0, c_0^2 \sigma^2)$. Therefore, since all c_i and σ^2 are all parameters to learn, the two definitions are identical.

Independently by the values of the parameters, a moving average process is always a weakly stationary process. It has a covariance function depending only on the distance between two considered points and a constant mean and variance. The covariance function of a $MA(q)$ is:

$$\gamma(\tau) = \begin{cases} \sigma^2 \sum_{i=0}^{q-\tau} c_i c_{i+1} & \tau \leq q \\ 0 & \tau > q \end{cases} \tag{2.23}$$

2.4.2. AR process

An Autoregressive process is a linear combination of the past values of the variable and of a white noise term.

Definition 2.4.2 (Autoregressive Process [13])**:** *Let $\eta \sim WN(0, \sigma^2)$ be a white noise, and let $a_1, a_2, ..., a_p \in \mathbb{R}$ be parameters. We call autoregressive process of order p, and we write $AR(p)$, a stochastic process defined as:*

$$y(t) = \sum_{i=1}^{p} a_i y(t-i) + \eta(t) \tag{2.24}$$

Similarly to the Moving Average process, it is possible to define the autoregressive process by summing to equation 2.24 a constant $\mu \in \mathbb{R}$.

$$y(t) = \mu + \sum_{i=1}^{p} a_i y(t-i) + \eta(t) \tag{2.25}$$

The second definition allows the process to have a mean different from zero. However, since the study of a zero mean process is simpler, I think it is better to select a definition having zero mean.

Differently from a moving average process, an autoregressive process may be non-stationary. To analyse when an autoregressive process is weakly stationary, we use the operator $zf(x) = f(x+1)$ (i.e., the operator is such that $z^{-1}f(x) = f(x-1)$). Therefore, the autoregressive process can be written as:

$$y(t) = \sum_{i=1}^{p} a_i z^{-i} y(t) + \eta(t) \tag{2.26}$$

Note that this notation allows expressing the autoregressive process by means of a transfer function, that is:

$$W(z) = \frac{1}{1 - \sum_{i=1}^{p} a_i z^{-i}} \tag{2.27}$$

This transfer function represents an asymptotically stable process if and only if all poles are strictly less than unitary in absolute value. In this case, an important result is that the autoregressive process tends asymptotically to a weakly stationary process [13].

The computation of the covariance of an autoregressive process is harder than that of

the covariance function of a moving average process. The computation of the covariance function at a given lag for an autoregressive process is given by the Yule-Walker equations. These equations serve both as a tool to compute the covariance function of a known process (known parameters) and as a tool to estimate the parameters of a process whose variance is known. The Yule-Walker equations are:

$$\gamma(\tau) = \sum_{i=1}^{n} a_i \gamma(\tau - i) \qquad \forall \tau \geq 1 \tag{2.28}$$

If the covariance function values are known for an $AR(p)$ process, it is possible to write a system of equations (Yule-Walker equations) to find the parameters of the process. Once the parameters have been found, the variance of the white noise term is obtainable through the process variance. Conversely, it is possible to compute the values of the covariance function if process parameters and the variance of the white noise are known.

2.4.3. ARMA process

An Autoregressive Moving Average process is a generalization of the moving average and autoregressive processes.

> **Definition 2.4.3** (ARMA Process [13])**:** *Let $\eta \sim WN(0, \sigma^2)$ be a white noise, let $c_0, c_1, ..., c_q, a_1, a_2, ..., a_p \in \mathbb{R}$ be parameters. We call ARMA process of order p, q, and we write $ARMA(p, q)$, a stochastic process defined as:*
>
> $$y(t) = \sum_{i=1}^{p} a_i y(t - i) + \sum_{i=0}^{q} c_i \eta(t - i) \tag{2.29}$$

A moving average process is a special case of an ARMA process in which $p = 0$, and an autoregressive process is a special case of an ARMA process in which $q = 0$.

As in the case of an autoregressive process, it is necessary to investigate when an ARMA process generates a weakly stationary process. To this end, the ARMA process can be

written using the delay operator (and so its transfer function):

$$y(t) = \sum_{i=1}^{p} a_i z^{-i} y(t) + \sum_{i=0}^{q} c_i z^{-i} \eta(t) \tag{2.30}$$

$$W(z) = \frac{\sum_{i=0}^{q} c_i z^{-i}}{1 - \sum_{i=1}^{p} a_i z^{-i}} \tag{2.31}$$

Since $W(z)$ is a transfer function, it represents an asymptotically stable process if and only if all its poles are less than unitary in absolute value. If this is the case, the ARMA process is weakly stationary [13].

2.4.4. ARIMA process

The definition of an Autoregressive Integrated Moving Average process is straightforward knowing the definition of an ARMA process. However, the differentiation of a time series is needed to define an ARIMA process. In the context of ARIMA processes, the differentiation of the time series refers to the process of computing a new time series by means of subtraction of the previous value to the current value. Namely, if $y(t)$ is a time series, its differenced time series (first order difference) is $y'(t) = y(t) - y(t-1)$. The second-order difference of the time series is the differentiation of the first-order differentiation of the time series. Therefore, we can express the differentiation as:

$$y^{(n)}(t) = y^{(n-1)}(t) - y^{(n-1)}(t-1) \qquad n \geq 1 \tag{2.32}$$

This definition of differentiation leads to an indirect definition of an ARMA process. It can be expanded as a summation of terms of the original time series up to $t - n$, where n is the order of the differentiation. In general, the differentiation of order n is:

$$y^{(n)}(t) = \sum_{i=0}^{n} k_i y(t-i) \qquad k_i \in \mathbb{R} \tag{2.33}$$

The definition of an ARIMA process as an ARMA process requires the exact expression of k_i terms. From the expansion of such terms, we can derive the following proposition:

Proposition 2.4.1: *Let $y(t)$ be a time series and $n \in \mathbb{N} \setminus 0$. The n^{th} order differentiation of the time series is:*

$$y^{(n)}(t) = \sum_{i=0}^{n} (-1)^i \binom{n}{i} y(t-i) \tag{2.34}$$

Proof. This proposition can be demonstrated by means of an inductive demonstration. The hypothesis and the thesis are:

$$Hp: \quad y^{(n)}(t) = y^{(n-1)}(t) - y^{(n-1)}(t-1) \wedge y^{(0)}(t) = y(t) \quad n \in \mathbb{N} \setminus 0$$

$$Th: \quad y^{(n)}(t) = \sum_{i=0}^{n} (-1)^i \binom{n}{i} y(t-i) \qquad \forall t$$

First, the base case is proved for $n = 1$.

$$y^{(1)}(t) = (-1)^0 \binom{1}{0} y(t) + (-1)^1 \binom{1}{1} y(t-1) = y(t) - y(t-1)$$

Therefore, the property is trivially true for $n = 1$. The proof requires it to be true for every $n+1$ when it is true for n.

$$\begin{aligned} y^{(n+1)}(t) &= \sum_{i=0}^{n+1} (-1)^i \binom{n+1}{i} y(t-i) \\ &= \sum_{i=0}^{n} (-1)^i \binom{n}{i} y(t-i) - \sum_{i=0}^{n} (-1)^i \binom{n}{i} y(t-1-i) \end{aligned}$$

The first right-hand term is the thesis, and the second is the hypothesis expanding the first and second terms using the thesis. The equivalence between the first right-hand term and the second right-hand term must be proved. If this is true, the thesis at $n+1$ can be reduced to the hypothesis at step n (the thesis has been used at step n on the hypothesis since it is assumed true), which we assumed to be true. We rewrite the three summations of the right-hand terms.

The first term is:

$$\sum_{i=0}^{n+1}(-1)^i \binom{n+1}{i} y(t-i) = y(t)+ \sum_{i=1}^{n}(-1)^i \binom{n+1}{i} y(t-i)+ + (-1)^{n+1} y(t-n-1)$$

The first summation of the second right-hand term is:

$$\sum_{i=0}^{n}(-1)^i \binom{n}{i} y(t-i) = y(t)+ \sum_{i=1}^{n}(-1)^i \binom{n}{i} y(t-i)$$

The second summation of the second right-hand term is:

$$\sum_{i=0}^{n}(-1)^i \binom{n}{i} y(t-1-i) = \sum_{i=0}^{n-1}(-1)^i \binom{n}{i} y(t-1-i)+ + (-1)^n y(t-1-n)$$

The first right-hand term should be equal to the subtraction between the second and the third right-hand terms ($x_1 = x_2 - x_3$). Therefore, we can simplify some terms and write:

$$\sum_{i=1}^{n}(-1)^i \binom{n+1}{i} y(t-i) = \sum_{i=1}^{n}(-1)^i \binom{n}{i} y(t-i)+ - \sum_{i=0}^{n-1}(-1)^i \binom{n}{i} y(t-1-i)$$

It can be observed that the last summation goes from 0 to $n-1$ while the others go from 1 to n. Therefore, it is possible to rewrite the equivalence as:

$$\sum_{i=1}^{n}(-1)^i \binom{n+1}{i} y(t-i) = \sum_{i=1}^{n}(-1)^i \binom{n}{i} y(t-i)+ - \sum_{i=1}^{n}(-1)^{i-1} \binom{n}{i-1} y(t-i)$$

Therefore, the minus in front of the last summation can be included in the power of term (-1).

$$\sum_{i=1}^{n}(-1)^i \binom{n+1}{i} y(t-i) = \sum_{i=1}^{n}(-1)^i \binom{n}{i} y(t-i) + \\ + \sum_{i=1}^{n}(-1)^i \binom{n}{i-1} y(t-i)$$

Now, and equivalence between the two summations can be written:

$$\sum_{i=1}^{n}(-1)^i y(t-i) \binom{n+1}{i} = \sum_{i=1}^{n}(-1)^i y(t-i) \left[\binom{n}{i} + \binom{n}{i-1} \right]$$

If the binomial coefficient on the left is equivalent to the binomial coefficient on the right, the thesis is proved to be reducible to the hypothesis at step n using the property at n.

$$\frac{(n+1)!}{i!(n+1-i)!} = \frac{n!}{i!(n-i)!} + \frac{n!}{(i-1)!(n-i+1)!}$$
$$\frac{(n+1)n!}{(i-1)!(n-i)!i(n+1-i)} = \frac{n!}{(i-1)!(n-i)!i} + \frac{n!}{(i-1)!(n-i)!(n-i+1)}$$
$$\frac{(n+1)}{i(n+1-i)} = \frac{1}{i} + \frac{1}{(n-i+1)}$$
$$\frac{(n+1)}{i(n+1-i)} = \frac{(n-i+1)+i}{i(n+1-i)}$$
$$n+1 = n+1-i+i$$

Then, since the thesis at step $n+1$ can be reduced to the hypothesis at step n, the proposition is true for any $n+1$ if the proposition is true at n. □

An $ARIMA(p,d,q)$ is defined as the process $ARMA(p,q)$ applied to the d^{th} order difference of the time series. The formal definition of an ARIMA process is:

Definition 2.4.4 (ARIMA Process)**:** *Let $\eta \sim WN(0,\sigma^2)$ be a white noise, let $c_0, c_1, ..., c_q, a_1, a_2, ..., a_p \in \mathbb{R}$ be parameters. We call ARIMA process of order p,d,q, and we write $ARIMA(p,d,q)$, a stochastic process defined as:*

$$y^{(d)}(t) = \sum_{i=1}^{p} a_i y^{(d)}(t-i) + \sum_{i=0}^{q} c_i \eta(t-i) \tag{2.35}$$

This process is helpful when the observed time series is not stationary. The differentiation of the data might produce a stationary time series (i.e., the differentiated time series is stationary), which is a desirable property for the data. Moreover, since the analytical expression of the n^{th} order difference of a time series is known, it is possible to derive the ARMA process generated by an ARIMA process.

Proposition 2.4.2: *Let $\eta \sim WN(0, \sigma^2)$ be a white noise, let $c_0, c_1, ..., c_q \in \mathbb{R}$ and $a_1, a_2, ..., a_p \in \mathbb{R}$ be parameters. The $ARIMA(p, d, q)$ is:*

$$\begin{aligned} y(t) = & -\sum_{i=1}^{n}(-1)^i \binom{n}{i} y(t-i) + \\ & + \sum_{i=1}^{p} a_i \sum_{j=0}^{n}(-1)^j \binom{n}{j} y(t-i-j) + \\ & + \sum_{i=0}^{q} c_i \eta(t-i) \end{aligned} \tag{2.36}$$

2.5. Forecasting time series

Many papers survey forecasting [52, 56, 77], the task of predicting future values based on past data. Let us say there are data up to $t_0 \in \mathcal{T}$ with $\mathcal{T}$ being a totally ordered set; the forecasting task consists in predicting values for $t \in \mathcal{T}$ such that $t > t_0$. The index t is generally strictly increasing and equally spaced. Some typical applications of time series forecasting are stock price forecasting and electrical load forecasting, which are of great interest for portfolio creation and evaluation and for energy suppliers. Time series forecasting is a well-known, widely developed, and widely explored research field. However, before the introduction of LSTM [40] and later of the Transformer [79], deep neural networks were not as efficient as statistical and machine learning approaches due to the problems of memorizing data. Even though these new blocks empower the capabilities of neural networks to forecast new values, the work in [57] shows that machine and deep learning do not outperform statistical approaches, and the work in [58] shows that machine learning approaches are now able to outperform statistical ones. In both competitions, they observe that the ensembles outperform single methods. This finding confirms that there is no superior approach for time series forecasting. However, the time series included in the competition are short (around 1900 points each) and numerous. Firstly, neural networks are slower to train with respect to ML and statistical approaches. Secondly, the bigger the network, the more data the network needs to learn a meaningful representation.

Consequently, submitting neural network models to these competitions might be infeasible due to time requirements. Nonetheless, the work in [87] makes similar observations: they present shallow neural networks capable of outperforming transformer-based neural networks for time series forecasting on publicly available datasets. Consequently, it seems that neural network advances are still immature to replace machine learning and statistical approaches used for forecasting and that even simple neural networks are enough to accomplish this task.

2.6. Anomaly detection

Anomaly Detection (AD) is an active research field of artificial intelligence and machine learning. Numerous papers [3, 5, 17, 19, 20, 23, 50, 67] surveyed it over the years from a general perspective and specifically for time series, the focus of this thesis. It is a particular instance of highly imbalanced binary classification where common practice identifies anomalous examples as the positive class and normal examples as the negative class. AD datasets are characterized by an infinitesimal portion of anomalies and a vast majority of normal points, making it practically impossible to adopt supervised learning solutions in most cases. Although supervised, unsupervised and semi-supervised solutions have been developed to tackle this problem in images, spatial, and temporal data, the last two techniques are broadly more common. The latter data type is spatial data ordered by some index t, usually named time. Because of that, it is called a data series or time series. It is central in many papers [4, 33, 41, 44, 48, 60, 76, 85] presenting their own AD dataset or benchmark. There are numerous datasets from different sources, one of which is cybersecurity. It is a natural field in which anomaly detection arises as a need naturally: fraudulent behaviour in banking, cyber attacks at power plants, and many others. The anomalous examples are usually related to a damaging or dangerous behaviour of the observed process; in some cases, it also follows permanent damage of a machine or permanent monetary loss. Therefore, detecting these behaviours is vitally important to avoid damage to people or to the company. This inherent imbalance between normal and anomalous classes makes the problem and the evaluation of models difficult. Notwithstanding the problems in collecting anomalous examples, there must be some anomalous points in the dataset; otherwise, algorithms can't be evaluated.

AD can be approached using several algorithms. There exist different directions by which it is possible to classify approaches: learning type (supervised or unsupervised), field (statistical, machine learning or deep learning), approach (prediction, reconstruction and others) and output (scores or labels). Many models have been proposed for all families

with different learning frameworks.

The **learning type** for an AD model is called semi-supervised: the model is learnt on normal data and tested on mixed data. Semi-supervised learning type is the current standard approach because of the previously discussed high class imbalance.

Even though models from all **fields** have been tested, there is a recent shift from statistical and machine learning approaches to neural network approaches. They offer many advantages: the most attractive is that deep neural networks do not need feature engineering since they extract features from data automatically.

The **ouput** of an AD model is generally of two types: score or label. The latter is a number which is 1 in case a point is considered an anomaly and 0 otherwise. The former is a score of abnormality of a point and is a more common way to report the output of the anomaly analysis. Scores can be bounded or unbounded and with high values for anomalies as well as low values for anomalies. A typical convention is to scale the score in $[0, 1]$, where 1 as the most anomalous score and 0 as the least anomalous score.

The **approaches** used to tackle AD are vast in number and in type. The work in [67] makes a well detailed description of the approaches used in AD. In short, they classify the approaches as:

- Forecasting-based methods are typically semi-supervised and forecast a number of points in the future, i.e., using points up to t, n points after t are predicted. The forecasted points are compared to the real ones: the residuals are the anomaly scores.
- Reconstruction-based methods are typically semi-supervised and reconstruct a subsequence of the time series of length: usually called **window**. A reconstruction approach consists receiving a portion of the time series (the window), mapping the window in a smaller space (optionally latent space), and mapping back from the smaller space to the original time series. The reconstruction error of a point (the difference between the input and the reconstructed sequence mapped back from the encoded space) is its anomaly score. However, it is vital to notice that if there are overlapping windows, there must be a method to score a point considering the multiple windows containing it (if two windows overlap, they partially include the same points). In general, reconstruction is the task of computing the original input from a smaller and different vector.
- Encoding-based methods are similar to reconstruction-based, with the difference that they score anomalies in the latent space instead of trying to reconstruct the

windows, e.g., they project the time series to a smaller space and compute the scores on that space.

- Distance-based methods are typically unsupervised and use a distance function to compute the distance of a point or subsequence to their neighbours. The higher the distance of a point or subsequence, the more anomalous.
- Distribution-based methods: can be either semi-supervised or unsupervised. They estimate a distribution over the data and classify points based on the learnt distribution.
- Isolation tree-based methods are typically semi-supervised. They build random ensembles of trees to partition data and isolate anomalies. The shorter the path to isolate a point, the more anomalous.

2.6.1. Historical methods: statistics

Statistical methods are either statistical models used to forecast time series or methods based on the computation and usage of statistics to determine which are the normal and the anomalous points. ARIMA [43] and its generalization SARIMAX is the most famous example of a statistical model usable for AD. It is the direct transposition of the ARIMA process to a model finding the parameters though fitting on data. It models the data generation process and forecasts points based on the learnt model. This model requires the search for the optimal orders to describe the time series, which are the so-called hyper-parameters. Although the autocorrelation and partial autocorrelation function helps identifying them, it might be that it is not suitable to model the time series. In fact, several other statistical models exist. Simpler models include the usage of thresholds based on mean absolute deviation and other statistical quantities.

2.6.2. Machine learning methods

Machine learning methods learn a model from data and generally require hand-crafted features. Most of the machine learning methods for AD have to be used in a semi-supervised or unsupervised manner. Some of the most common machine learning methods for AD are Isolation Forest [54], Local Outlier Factor [15], and One-class SVM [55]. Isolation Forest learns to isolate samples: the shorter the average tree path to isolate the sample, the more anomalous. Local outlier factor learns to classify a point as anomalous or not based on the distance to its neighbours. One-class SVM learns the support vectors identifying the normal class of data and classifies new points based on their position with

respect to the learnt surface. Clustering approaches (such as K-Means) cluster points and compute the distance between points and their centroid as anomaly scores [67].

Machine learning approaches for AD on time series tend to use sliding windows [23, 67] to incorporate time information. A sliding window has two parameters: the shift and the window length. The window length is the number of contiguous points inside the window. The shift represents the number of points by which it is moved to the right on the time series while producing vectors (like the strides parameter of convolutional layers in neural networks). Let $X_n \in \mathbb{R}^f$ be a time series with n points and f features. The sliding window produces vectors in a space $\mathbb{R}^{w \times f}$ where w is the length of the window. Precisely, the sliding window space is $W_{n_w} \in \mathbb{R}^{w \times f}$ where $n_w = \lfloor \frac{n-w}{s} \rfloor + 1$ is the number of windows and s is the shift. Let $w_i \in W_{n_w}$ and $x_i \in X_n$, the sliding windows are:

$$w_i = [x_{is}, x_{is+1}, ..., x_{is+w-1}] \tag{2.37}$$

The windows are matrices, and most machine learning methods work on vectors. Therefore, when the time series is multivariate, it could be necessary to flatten the matrix to a vector. A possibility is to put all features in order such that the new window is:

$$w_i = [x_{is,1}, ..., x_{is+w-1,1}, x_{is,2}, ..., x_{is+w-1,f}] \tag{2.38}$$

However, windows will grow fast using such an approach. Although another option is to analyse channels separately and group together the results, [73] observes that a time series should be considered an entity instead of analysing the channels separately. A feature may be correlated with others to some degree, thus giving the contextual information that would not be present analysing channels separately.

After the time series has been projected in the window space, it can be analysed and used for training. Nonetheless, it is critical to define the way in which the scores or predictions are handled in case of overlapping windows. When the shift is less than the window size ($s < w$), windows will share some points. Consequently, points might be reconstructed, forecasted or assigned to a score multiple times. In each case, it is necessary to decide which strategy to use to give a final score to the point. For reconstruction and forecasting, there are several options:

- Median: take the median forecasted or reconstructed value (e.g., [34] takes the median value for the reconstruction part).
- Mean: take the mean forecasted or reconstructed value.

In purely spatial approaches that do not provide a reconstruction or a forecast, each window will have a label or a score. If the model assigns a label without any score, a voting scheme is used to obtain the labels of points. If the windows have a score, there exist several options to propagate the score from the overlapping windows to the point:

- Median: take the median score.
- Mean: take the mean score.
- Minimum: take the minimum score (the idea is that overlapping windows containing an anomalous point will have higher scores).
- Maximum: take the maximum score.

2.6.3. Neural approaches

Neural approaches to AD are more and more common. Several architectures have been used successfully to identify anomalies in datasets. Lately, approaches using transformers [78, 86], GANs [34], variational autoencoders, convolutional, mixes of the previous [21, 73, 88] and others [18, 30] have been proposed. This section limits itself to introducing the general usage of neural networks for anomaly detection in time series. Networks must learn temporal dependencies between time points as well as machine learning. Also, neural networks use sliding windows to encode information of close timestamps to predict the future, but there are a few differences. For instance, neural networks tend to be either predictive (forecast some points in the future) or reconstructive. With reconstruction approaches or GANs, windows will have a reconstruction and possibly a critic score [34] (only with GANs, the output of the critic net distinguishing how the input seems real or not). Since such values describe the windows and not the points, it is important to decide how to give scores to points using either the median or the mean for the reconstruction. The work in [34] applies kernel density estimation to the collection of critic scores assigned to a point and takes the maximum smoothed value. For forecasting approaches, it is important to note that even if the windows overlaps, each point might have only one prediction; e.g., if the forecasting length is 1, each point is predicted only once. Therefore, points will have multiple forecasted values if the forecasting length is greater than 1, and if the shift is smaller than the forecasting length. In such a case, it is important to state whether the forecasted value for a point is the mean or the median of all the forecasted values. However, it is not necessary to define a reconstructed value or a forecasted value for a point if only anomaly scores are in the interest of the approach. The work in [59] proposes to estimate a Gaussian distribution of the errors on the normal data points of

the validation set. The error vectors used to estimate the distribution are:

$$e(t) = [e_{11}(t), ..., e_{1l}(t), ..., e_{d1}(t), ..., e_{dl}(t)] \tag{2.39}$$

Where $e(t)$ is the error vector for the t^{th} point of the time series, each feature is predicted (or reconstructed) l times, and d features are predicted (or reconstructed). Suppose for a moment that the approach is a forecasting-based one (the reasoning is similar for a reconstruction-based), $e_{ij}(t)$ is the error between the real feature i at time t and the predicted feature i at time t predicted at time $t-j$; e.g., the feature $x_i(t)$ will be predicted l times; therefore, there will be l differences for that feature. It is not mandatory to predict (or reconstruct) all the features of a time series. Similarly, for reconstruction-based approaches, j identifies the reconstructed feature by the window at time $t-j$.

2.7. Current ML and AD libraries

Creating machine learning models from scratch is a long, time-consuming and complex task. Because of this complexity, many libraries have been developed and made available to the scientific community. Machine learning solutions are developed in several languages. However, Python is one of the most (if not the most) used programming languages for machine learning development because of its simplicity and intuitiveness. Besides its numerous positive sides, it has some drawbacks:

- Speed: it is slow since it is interpreted and offers helpful features such as duck typing.
- Dynamic typing: types are dynamically checked, which is effective, but sometimes draws away the attention of developers creating the constant need to verify the documentation.

Given the need for speed in machine learning and the simplicity of Python, several tools have been developed to perform numerical operations. Since it allows the creation of libraries written in C to expose a Python API, libraries such as numpy [39], numba [47], pandas [63, 84], scipy [81], scikit-learn [65], pytorch [64], tensorflow [2] and others have been developed to perform numerical, and machine learning tasks fast in Python. These libraries greatly aid the data scientist in the development of machine learning solutions. However, libraries such as [2, 65] are generic for machine learning and are not specific for time series. Libraries like statsmodels [68] add functionalities to analyse and create models specifically designed for time series. Since time series anomaly detection is an extremely specific task, there are few libraries for it, and even few are of good quality. Time series

anomaly detection includes some specific tasks that are almost always present, and very few libraries include modules designed to aid the development of specific functionalities of the anomaly detection methods. Two valid libraries addressing the problem are Orion [7], and PyOD [89]. The latter is a library implementing several anomaly detection methods. The former is a library based on the machine learning library MLPrimitives/MLBlocks [69, 80] completely based on the concept of a pipeline for time series. Some of these libraries will be discussed and compared to the proposed anomaly detection library in chapter 4.

3 | Anomaly detection evaluation

As with any other machine learning task, AD methods must be evaluated against a performance measure. Since AD is an instance of imbalanced classification with a skewed distribution of labels towards the normal class, different imbalanced classification metrics are appropriate for AD. Even though AD datasets are labelled (if the anomalies are unknown, how could an algorithm's quality be assessed?), the training often follows a semi-supervised approach (only normal points in the training).

This chapter describes the evaluation of anomaly detection methods and datasets. Section 3.1 briefly introduces general machine learning metrics. Section 3.2 introduces imbalanced classification metrics. Section 3.3 presents the metrics used in AD, along with the advantages and disadvantages. Section 3.4 analyses the bias in publicly available datasets and proposes an automatic approach to evaluate their inherent simplicity to enrich the critical analysis of anomaly detection methods on public datasets.

3.1. Metrics used in machine learning

Machine learning is useful for solving several problems (like increasing the accuracy of a classification system). It aims to create a model from data to minimize (or maximize) a performance measure called loss function (or utility function) [11]. Moreover, numerical quantities used to evaluate the model are also called metrics. The key difference between the two numerical quantities is that the former is used for finding the parameters during training, whereas the latter is used only for evaluation. Different tasks have different objectives and metrics, e.g., accuracy cannot be used for regression. An example of a loss function is the Mean Squared Error (MSE) in regression problems:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3.1}$$

Where n is the number of points, y_i is the true value and $\hat{y}_i$ is its prediction. The objective of this loss function is to find the model whose predictions are the most similar

to the real values. It has two desirable properties for loss functions: continuity and differentiability. Since the global optimization of a function that is not differentiable and not even continuous is hard, many current global optimization algorithms assume continuity; e.g., differential evolution [72] assumes the function is continuous, SHGO [28] assumes the function is continuous and works better if it is Lipschitz smooth, and DIRECT [45] is a Lipschitzian optimization algorithm (assumes Lipschitz continuity).

Each task has specific loss functions to accomplish the final result. A common loss function for classification is the Categorical Cross Entropy (CCE) or the Sparse Categorical Cross Entropy (SCCE). They are the same loss function, and they differ only in terms of label format. The CCE for a k class classification problem has 1-of-K coding scheme vector for the target (all zeros and 1 in the position of the correct class) [11], while the SCCE has integers as targets. Let φ_i be the feature vector of point i, t_{ij} the j^{th} element of the target vector of point i, and $\hat{y}_{ij}$ the predicted probability that the point i is of class j. The CCE is defined as:

$$CCE = -\sum_{i=1}^{N}\sum_{j=1}^{K} t_{ij}\ln(y_{ij}) \tag{3.2}$$

This loss function aims at giving probability 1 to the correct class for each point. Indeed, it is the negative log-likelihood function for the multi-class classification problem. However, it does not take into account imbalances in data.

Besides giving an introductory discussion on the loss functions that could be used for regression or classification, it is worth mentioning some of the metrics used for classification tasks since this thesis is about AD (a specific instance of imbalanced classification). Let TP, TN, FN, FP be true positives, true negative, false negatives, and false positives; Accuracy (Acc), Precision (Pre), Recall (Rec), True Positive Rate (TPR) and True Negative Rate (TNR) are defined as:

$$\text{Acc} = \frac{TP + TN}{TP + FN + TN + FP} \tag{3.3}$$

$$\text{Pre} = \frac{TP}{TP + FP} \tag{3.4}$$

$$\text{TPR} = \frac{TP}{TP + FN} \tag{3.5}$$

$$\text{Rec} = \text{TPR} \tag{3.6}$$

$$\text{TNR} = \frac{TN}{TN + FP} \tag{3.7}$$

3.2. Metrics used in imbalanced classification

Imbalanced classification consists of labelling objects into one of K classes over a dataset whose classes have different proportions. Contrariwise, balanced classification over K classes have n/K points for each class (same proportion). Therefore, the CCE is an appropriate loss for balanced classification since each point has the same importance (expressed as the implicit weight of one). Conversely, if classes have different population sizes and thus it is impossible to train the method on a set of classes with the same proportion, it is possible to employ a weighted loss function to improve the quality of learning. An option is to use Weigthed Categorical Cross Entropy (WCCE):

$$WCCE = -\sum_{i=1}^{N}\sum_{j=1}^{K} w_j t_{ij} \ln(y_{ij}) \tag{3.8}$$

Where w_j is the weight of class j. There are several other ways to deal with imbalances, such as modifying the algorithm to consider the imbalance [32, 46, 71]. Besides the training, there are many metrics to evaluate imbalanced classification. One of which is the Balanced Accuracy (BAcc), a weighted version of the Acc removing the bias towards the majority class [36].

$$\text{BalancedAccuracy} = \frac{1}{K}\sum_{i=1}^{K}\frac{c_{ii}}{\sum_{j=1}^{K} c_{ij}} \tag{3.9}$$

Where c_{ij} is the number of elements of class i that are predicted as being of class j.

3.3. Metrics used in anomaly detection

Being AD an instance of imbalanced binary classification, the reader might think that metrics for imbalanced binary classification are all appropriate in anomaly detection; this is not true in practice. Since anomalies describe undesired and optionally dangerous or faulty behaviour, it is imperative to identify all anomalies and avoid the misclassification of normalities. However, many models only provide a score of abnormality for each point, which must be thresholded to obtain labels, i.e., the detections. Such problems are orthogonal [67]. Chaining the two approaches and evaluating classification metrics might underestimate the goodness of the model (e.g., the thresholding method is poor). To evaluate efficacy of detection methods (label as output), TPR and FPR (also called false alarm rate, since anomalies should raise alerts in monitoring contexts) are useful [50]. If TPR and FPR are optimal (1 and 0), the algorithm yields perfect detection. Another

widely used measure to evaluate detections (for binary classifications) is F_β Score (F_β); it is a weighted harmonic mean [31] between Pre and Rec [70]:

$$F_\beta = \frac{(1+\beta^2)\text{pre}\cdot\text{rec}}{\beta^2\text{pre}+\text{rec}} = \frac{(1+\beta^2)TP}{(1+\beta^2)TP+\beta^2FN+FP} \tag{3.10}$$

Where TP, FN, FP mean true positives, false negatives, and false positives. The most common value for F_β is $\beta = 1$. Besides using metrics to measure the goodness of detections, there are also metrics to evaluate the quality of the scores, i.e., the ability of the model to separate normal and anomalous classes. The most common measures for evaluating scores are the Area Under the Curve (AUC) measures; e.g., Area Under the Receiver Operating Characteristics Curve (AUC-ROC) and Area Under the Presicion-Recall Cruve (AUC-PR) are two metrics used in [67].

Besides the previous standard metrics to evaluate AD and classification, papers proposed metrics to evaluate range-based detections and online detections. The work in [75] proposes a variation of the point-based Pre and Rec metrics designed to work with ranges rather than with points; therefore, they propose range-based metrics. Regarding online detections, [49] defines a metric for the evaluation of online detections called NAB score. However, the usage of these metrics is not a standard practice for the moment.

3.3.1. Metrics used by current state-of-the-art papers

The anomaly detection literature is vast, and there is not common agreement on the performance measures to use. Here there is a list of some of the latest works in AD:

- The works in [78, 86] use both F_β and AUC-ROC.
- The works in [34, 88] report F_β with uncertainty.
- The work in [73] reports F_β.
- The work in [18] reports the best F_β with uncertainty (only for their model) selecting the threshold on the test set with adjustment of predicted labels (i.e., if a point in an anomalous interval is labelled as anomalous, all points in the interval are considered true positives besides what is the output of the network).
- The works in [21, 30] report the best F_β on the test set.

Even though all the previous measures are appropriate for anomaly detection, there might be some cases in which some measures should be avoided or coupled with the usage of other measures. The choice of the performance measure depends on the aim of the evaluation:

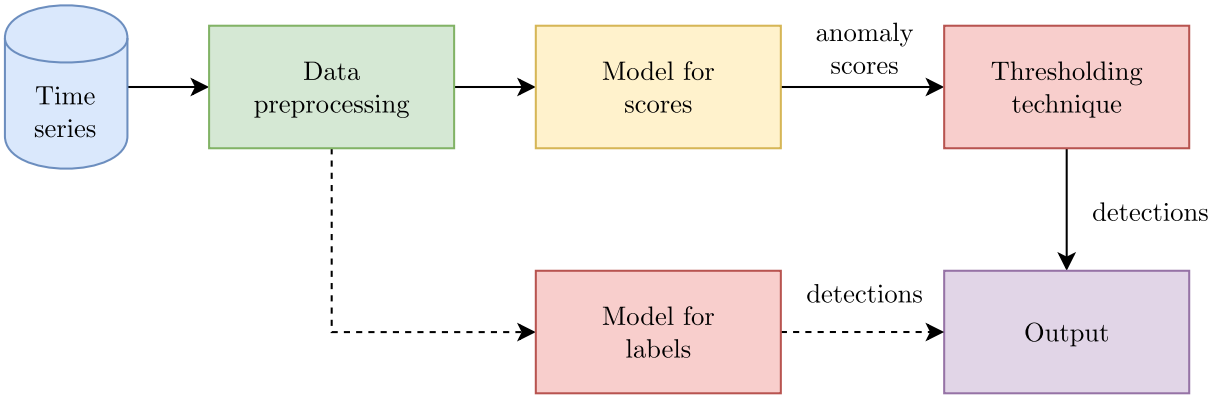


Figure 3.1: It is the flowchart of the two possible ways to create a model for AD: use a model giving scores to points and subsequently convert scores into labels, or use a model directly outputting labels.

the model's ability to separate normal and anomalous classes, the capability of a new thresholding mechanism to provide decent thresholds, or both together.

3.3.2. How to choose the metric

The choice of the metric for evaluation purposes depends on whether scores or detections are evaluated. Threshold-agnostic measures like AUC-ROC and AUC-PR are appropriate for analysing the capability of a model to separate normal and anomalous classes. Threshold-dependent measures like F_β, Pre, Rec, TPR and FPR are serviceable for analysing the quality of detections. Assuming both scores and detections are under evaluation, both categories are needed for the assessment, e.g., AUC-ROC for evaluating scores and F_β for evaluating detections. A notable example of the latter case is the works in [78, 86], they provide both the F_β and AUC-ROC.

Computing the best obtainable threshold-dependent measure (like F_β) on the test by varying the threshold τ to transform scores into labels is technically correct because it returns a threshold-agnostic measure. Moreover, it substitutes the role of the thresholding mechanism (see figure 3.1) of converting scores into labels with a method solely used for the computation of metrics that require detections, but only scores are available. This means that the proposed solution is incomplete and cannot be used in practice since it misses the thresholding component to transform scores into labels. Therefore, this numerical quantity is of little interest for some reasons:

- It is a purely theoretical result: such a score might be unobtainable since the thresholding mechanism must be unsupervised or semi-supervised.

- Change of semantics: it transforms a threshold-dependent metric into a threshold-agnostic metric providing. Moreover, it might produce the exact same final information as other threshold-agnostic metrics.

Therefore, using a threshold-agnostic measure is natural for scores and gives the same information. If both labels and scores of different models are under evaluation (some models output a label and some output a score for a point), the adoption of a best obtainable threshold-dependent measure for scores is a good choice. These measures are able to compare the detection power of scores against raw detections by assuming that the underlining threshold mechanism is perfect.

Solely reporting the threshold measures such as F_β when both a model and a thresholding algorithm are presented hides the information on the contribution of each method to the final result. Low scores might be mainly due to the thresholding method being inappropriate. This practice hinders the complete and simple understanding of the results.

Finally, in each case, the confidence on the performance measures should be reported with all the details needed to reproduce such values (such as seeds). Avoiding reporting confidence disallows the reader to understand the degree of the stability of the model. If one model has a minuscule variation and another has an immense variation, even if the method with immense variation has a slightly higher mean, it could be discarded in favour of the more stable one. That is, tighter confidence intervals are preferred over larger confidence intervals.

3.4. Bias in datasets and annotations

To the best of my knowledge, [85] is the only paper approaching the problem of datasets' inherent triviality for anomaly detection. They claim that some datasets are simple and solvable through simple approaches. Therefore, it is not of interest to evaluate complex and hardly explainable models such as neural networks on them, and the research community should abandon them. Given this interesting and important statement, a question arises: is it possible to define a measure of triviality and an algorithm computing such a measure? In their work, [85] define a dataset as simple if there exists a one-liner solving the dataset, i.e., a method writable in one line of code achieving Acc = 1. They also give an example of a one-liner that might be used to evaluate the triviality:

$$x > c_1\text{movmean}(x, w) + c_2\text{movstd}(x, w) + b \tag{3.11}$$

Where x is either the time series, the differentiation of the time series (see Section 2.4.4) or the absolute value of the differentiation of the time series, $w \in \mathbb{N}$ is the sliding window length, $b \in \mathbb{R}$ is a constant, c_1 is a constant which is either 0 or 1, and $c_2 \in \mathbb{R}$ is a constant. The moving average is defined as in definition 2.3.4, and the moving standard deviation is defined using the same sliding window approach. The only difference with the previous definition is that values outside the time series are considered to be 0 such that it returns a series of the same length as the input series. The moving average performs a smoothing of the time series, the standard deviation measures the sparsity in the subsequence, and the constant adjusts the series for correct detection. Figure 3.2a shows an example of the approach presented in [85]. The reader should observe that values lower than the moving average will never be detected as anomalies. An improvement of their method could be to consider in parallel also the following inequality:

$$x < c_1 \text{movmean}(x, w) - c_2 \text{movstd}(x, w) - b \tag{3.12}$$

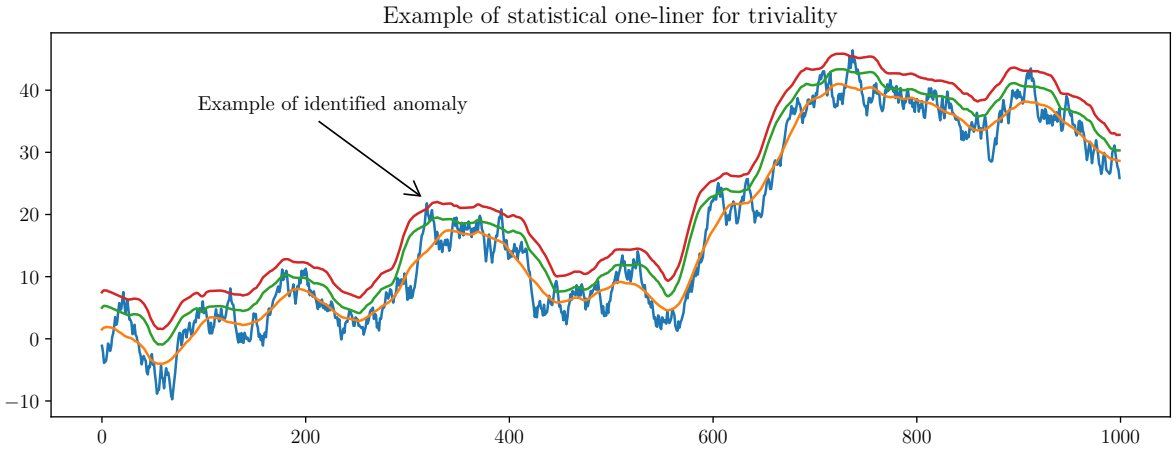
In which the constants are the same as those in equation 3.11. Therefore, a system of the equations 3.11 and 3.12 will detect abnormally low or high values as anomalies. However, a statement against that improvement could be that the first difference of a time series tends to be stationary. Thus the time series depicted in figure 3.2 is not realistic.

To clear doubts about the correctness of the example in figure 3.2, figure 3.3 shows two time series. The time series in figure 3.3a is a time series whose first order differentiation generates the time series in 3.3b, which is the time series used in 3.2. Therefore, even smooth and rather simple functions might generate complex and non-stationary time series after differentiation. Moreover, even if this is uncommon behaviour, it is still an example of a case not covered by the previous approach.

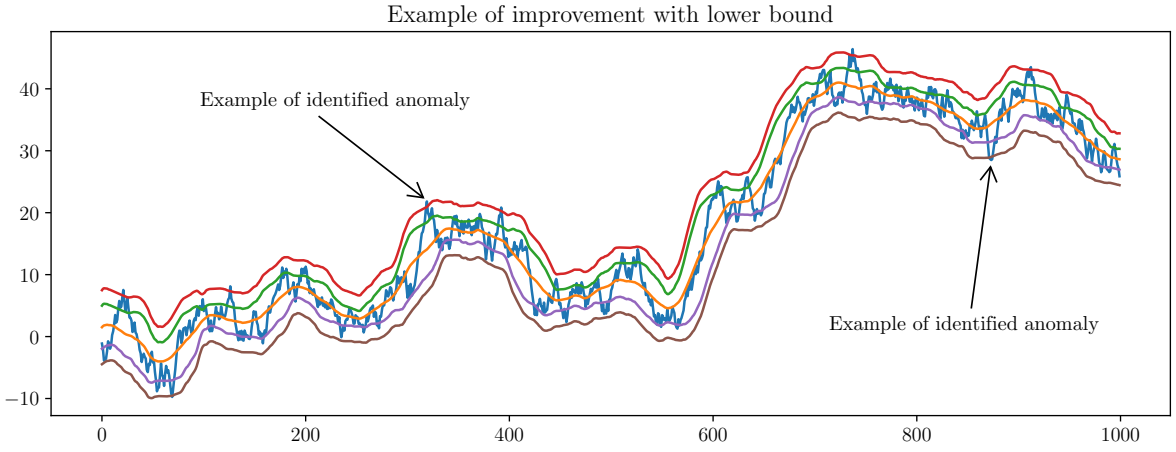
Therefore, given the problems reported by [85] on some publicly available datasets regarding their simplicity, there is the need for an automatic tool evaluating the degree of simplicity of datasets for AD.

3.4.1. Simple datasets

The task of evaluating whether a dataset is simple is partially independent of the task of learning to solve simple datasets. A dataset might have simple spatial and temporal characteristics hardly identifiable from methods. Therefore, I propose an approach to evaluate the simplicity of a dataset based on its spatiotemporal characteristics. Moreover, since an AD dataset is a classification dataset, a sort of simplicity definition already exists: linear separability [11].



(a)

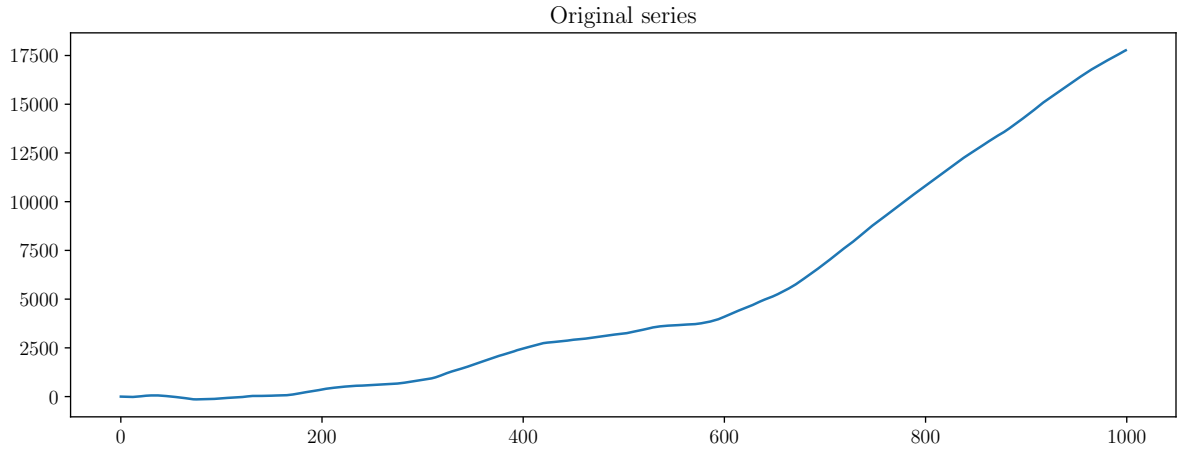


(b)

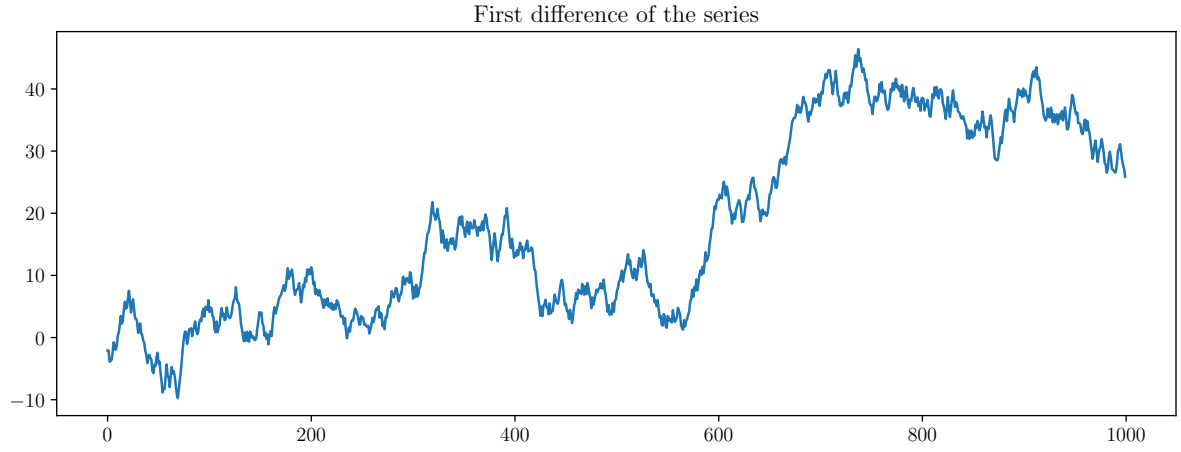
Figure 3.2: One-liner approach proposed by [85] and an improvement having $c_1 = 1$, $c_2 = 1$ and $b = 2.5$. Blue is the time series, orange is the moving average, green is the sum of the moving average and moving standard deviation, red is the complete right-hand side, violet is the moving average to which the moving standard deviation is subtracted, and brown is the complete right-hand side subtracting the constant and the moving standard deviation. (a) the one-liner exactly defined in [85]. (b) the enhancement of the one-liner to get also anomalies of abnormally low values. The images can be generated by running the script "example_wu_approach.py".

> *«Data sets whose classes can be separated exactly by linear decision surfaces are said to be linearly separable.»* — Christopher M. Bishop

Linear separability is a property which makes a classification dataset simple. The *perceptron convergence theorem* supports this strong statement: the perceptron algorithm is



(a)

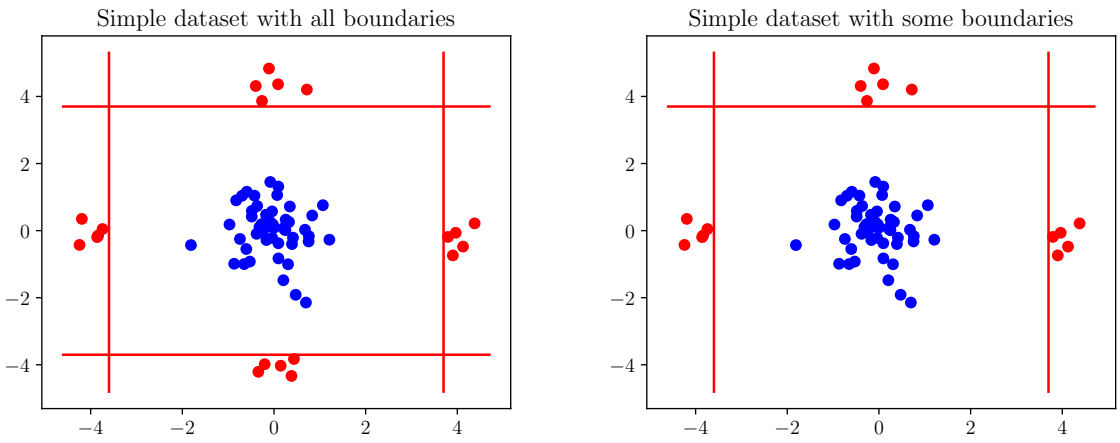


(b)

Figure 3.3: The time series in (a) generates the time series in (b) by first order differentiation. The images can be generated by running the script "example_wu_approach.py".

guaranteed to find an exact solution in a finite number of steps if the dataset is linearly separable [11].

It is possible to define notions of simplicity for anomaly detection resembling the linear separability property. It is known in the AD community that point anomalies [23] are simple to solve, and contextual and collective anomalies [23] are harder. Since point anomalies are typically out-of-range points, they should be detectable by comparison against a constant (or constant vector for multivariate time series). Contextual and collective anomalies need a way to embed time information to be detected. A possible approach is to use a sliding window of length w, thus incorporating the temporal knowledge up to $t \pm \frac{w}{2}$. However, since the aim is to evaluate the simplicity of a dataset, the idea is to use



(a) (b)

Figure 3.4: Simple datasets with 2 dimensions in which (a) has comparison constants on all dimensions, and (b) has comparison constants on some dimensions. The images can be generated by running the script "example_simplicity.py". Red points are anomalies, blue points are normal points.

only statistical quantities over the sliding window. Specifically, the moving average and moving standard deviations are used to define a simplicity score. The moving average is used to analyse the change in mean, and the moving standard deviation is used to analyse the variability of points. Aiming at incorporating a simplicity definition for AD, I define simplicity as:

> **Definition 3.4.1** (Dataset simplicity)**:** *Let $\overline{x} \in \mathbb{R}^{n \times f}$ be a time series with n points and f features. The time series $\overline{x}$ is said to be simple if for all $1 \leq i \leq f$, $1 \leq k \leq f$, and $1 \leq j \leq n$, there exist constants $c_i^u \in \mathbb{R}$ and $c_i^l \in \mathbb{R}$ such that the following rules for the identification of anomalies provide $Acc = 1$:*
>
> $$\begin{aligned} \overline{x}_{j,i} &\leq c_i^l \\ \overline{x}_{j,k} &\geq c_k^u \end{aligned} \tag{3.13}$$

Namely, the definition states that if there are constants on one or multiple dimensions of the time series separating the anomalous and normal class, the time series is simple. Figure 3.4a is an example of a time series scatter plot in which anomalous and normal points are perfectly separable through constant comparison (observe that there is no temporal information). The analysis of simplicity does not consider temporal dependencies (and a time series is a set of vectors ordered by an index); it only considers spatial values of

the points. Since the time series is seen as a spatial dataset to analyse simplicity, this problem is analogous to finding the minimum bounding box of a set of points.

Furthermore, the aim is to obtain a score of simplicity from the definition of a simple dataset: the definition 3.4.1 must be translated into a formal definition of a score of simplicity. Therefore, by leveraging the use of constants, it is possible to define the following score:

Definition 3.4.2 (Constant simplicity score)**:** *Let $\overline{x} \in \mathbb{R}^{n \times f}$ be a time series with n points and f features. The constant simplicity score of the time series $\overline{x}$ is defined as* $\max(TPR)$ *@* $TNR = 1$*, of the following function, where @ means "at the value of":*

$$f(i, j, c_j^l, c_j^u) = \begin{cases} 1, & \text{if } \overline{x}_{i,j} \leq c_j^l \vee \overline{x}_{i,j} \geq c_j^u \\ 0, & \text{otherwise} \end{cases} \tag{3.14}$$

Where c_j^l, c_j^u are constants. That is, the highest percentage of anomalous points separable from normal points without producing false positives.

A straightforward definition leverages the fact that there is no interest in simple anomalies; complex anomalies enable the development of the scientific process.

If a dataset has a high constant simplicity score, a high percentage of anomalies can be separated from normal points by exploiting the spatial properties of the time series. If the score is one, anomalies and normalities can be perfectly separated only using spatial properties. Therefore, the temporal information is useless for the identification of anomalies. A time series whose temporal information is useless for the task is of limited interest to the scientific community. However, if the anomalies can be found on the moving average or moving standard deviation series, the series is simple too. Accordingly, the following scores can be defined:

Definition 3.4.3 (Moving average simplicity score)**:** *Let $\overline{x} \in \mathbb{R}^{n \times f}$ be a time series with n points and f features, and $w \in \mathbb{N} \setminus \{0\}$. The moving average simplicity score of the time series $\overline{x}$ is the constant simplicity score of the series* $\text{movmean}(\overline{x}, w)$.

Definition 3.4.4 (Moving standard deviation simplicity score)**:** *Let $\overline{x} \in \mathbb{R}^{n \times f}$ be a time series with n points and f features, and $w \in \mathbb{N} \setminus \{0\}$. The moving standard deviation simplicity score of the time series $\overline{x}$ is the constant simplicity score of the series* $\text{movstd}(\overline{x}, w)$.

These scores can give an idea of the complexity of a time series. If any of the scores is maximum (all scores are in the interval $[0, 1]$), anomalies can be perfectly separated from normal points. However, if all scores are in the interval $(0, 1)$, it might be possible that the dataset is simple in general or simpler than what single scores suggest: different scores might separate distinct anomalies. A more general score is defined to capture anomalies by all the previous scores:

Definition 3.4.5 (Mixed simplicity score)**:** *Let $\overline{x} \in \mathbb{R}^{n \times f}$ be a time series with n points and f features, and $w \in \mathbb{N} \setminus \{0\}$ be a constant. The mixed simplicity score of the time series $\overline{x}$ is defined as* $\max(TPR)$ *@* $TNR = 1$*, of the following function, where @ means "at the value of":*

$$f(i, j, c^l_{j,v}, c^u_{j,v}) = \begin{cases} 1, & \text{if } \overline{x}_{i,j} \leq c^l_{j,1} \vee \overline{x}_{i,j} \geq c^u_{j,1} \\ 1, & \text{if } \text{mvavg}(\overline{x}, w)_{i,j} \leq c^l_{j,2} \vee \text{mvavg}(\overline{x}, w)_{i,j} \geq c^u_{j,2} \\ 1, & \text{if } \text{mvstd}(\overline{x}, w)_{i,j} \leq c^l_{j,3} \vee \text{mvstd}(\overline{x}, w)_{i,j} \geq c^u_{j,3} \\ 0, & \text{otherwise} \end{cases} \tag{3.15}$$

Where $c^l_{j,v}, c^u_{j,v}$ are constants. That is, the highest percentage of anomalous points separable from normal points without producing false positives.

Given the definitions of scores, it is possible to introduce algorithms to compute them. The first algorithm computes the constant simplicity score since it is the basis for other algorithms. The constant simplicity algorithm pseudo-code (algorithm 3.1) has worst-case time complexity $T(N, F, A) = NF\log(N) + ANF + NF + N = \Theta(NF\log(N) + ANF)$ where N is the number of points in the input time series, A is the number of anomalous points, and F is the number of features in the input time series. Here I list all the passages in a bulleted list:

- The ordering of the time series costs $\Theta(NF\log(N))$, and the initialization of lower and upper bound lists costs $\Theta(F)$.
- The outer loop costs $\Theta(F)$ since it iterates over all the channels of the time series.

- The inner loop executes at most $\Theta(AN)$. The TPR increases until there are anomalies separable from normalities, and since the TNR never decreases, the cycle costs $\Theta(A)$. Inside the loop, the computation of TPR and TNR requires $\Theta(N)$ since the comparison against a constant and the count of positive (or negative) rate can be parallelized.
- After the loop, the labels can be obtained with $\Theta(NF)$ since every feature is compared with the lower and upper bound. Finally, the TPR between two vectors of length N costs $\Theta(N)$.

Differently, the best case time complexity is $T(N, F, A) = NF\log(N) + 2NF + N = \Theta(NF\log(N))$. It is the case in which the constant simplicity score is 0; the TNR will decrease at the first iteration, and the `break` will be executed. Therefore, the time complexity is controlled by the ordering of the time series.

The pseudo-code for the moving average and moving standard deviation code is almost identical. The only difference is that the former calculates the average over the windows, and the latter computes the standard deviation. The algorithm 3.2 defines both the algorithm to compute the moving average simplicity score and a heuristic to obtain a list of window lengths to try while searching the moving average simplicity score. The exact moving average score would require trying all the possible windows. However, with long windows, adding a few other points will slightly change the mean; therefore, a logarithmic increment is chosen. The pseudo-code for computing the moving standard deviation simplicity score differs from it only at line 25, in which "movmean" is substituted with "movstd".

Given the previous algorithms, the pseudo-code of the mixed score is straightforward. It calls all the the previous simplicity algorithms, computes the labels of each approach and merges them by calculating the OR between the labels; i.e., if a point is identified as an anomaly by at least one of the previous, it is identified as an anomaly.

Finally, figure 3.5 shows the previously introduced scores for some publicly available datasets. It is possible to observe that Yahoo Webscope S5 [48] has a substantially high constant simplicity score. Indeed, it contains several point anomalies in the first three benchmarks (A1, A2, A3). Other datasets containing fewer point anomalies have a lower constant simplicity score (such as UCR and NAB), which increases with moving average and standard deviation simplicity scores. Several datasets have anomalies separable from normal points, and datasets like UCR [85] have complex and simple series. Differently, MGAB [76] and GHL [33] have anomalies which are not separable from normal points just by looking at these simple statistics. Therefore, most benchmarks contain both complex

Algorithm 3.1 Constant simplicity score

```
procedure COMPUTE-CONSTANT-SCORE(x̄, labels)
    x_asc = order-by-feature(x̄, "asc")
    lower, upper = each is list of num_features NaN
    for i = 0 to num_features do
        tnr_up, tpr_up, score_up = 1, 0, 0
        tnr_dw, tpr_dw, score_dw = 1, 0, 0
        for j = 0 to num_points do
            if tnr_up == 1 then
                tpr_up = tpr(x̄[all, i] ≥ x_asc[num_points - 1 - j, i], labels)
                tnr_up = tnr(x̄[all, i] ≥ x_asc[num_points - 1 - j, i], labels)
                if tpr_up > score_up and tnr_up == 1 then
                    upper[i] = x_asc[num_points - 1 - j, i])
                    score_up = tpr_up
                end if
            end if
            if tnr_dw == 1 then
                tpr_dw = tpr(x̄[all, i] ≤ x_asc[j, i], labels)
                tnr_dw = tnr(x̄[all, i] ≤ x_asc[j, i], labels)
                if tpr_dw > score_dw and tnr_dw == 1 then
                    lower[i] = x_asc[j, i])
                    score_dw = tpr_dw
                end if
            end if
            if tnr_up != 1 and tnr_dw != 1 then
                break
            end if
        end for
    end for
    labels_pred = compare x̄ with lower and upper
    score = tpr(labels_pred, labels)
    return score, lower, upper
end procedure
```

and simple time series. If a study wants to pick series at random from a benchmark, it should either devote attention to the series that are being chosen or use fully complex datasets. However, since I want to emphasize the importance of reproducing and comparing results, I suggest testing on the overall benchmark while evaluating a method.

3.4.2. Simple datasets vs. simple methods

This section briefly discusses the difference between the analysis of the intrinsic simplicity of a dataset and the complexity of an AD method. Simple approaches may detect anoma-

Algorithm 3.2 Moving average simplicity score and window choice

```
procedure GET-WINDOWS(range)
    windows, i = empty list, range[0]
    while i ≤ range[1] do
        windows.append(i)
        if i < 5 then
            i += 1
        else if 5 ≤ i < 20 then
            i += 5
        else if 20 ≤ i < 100 then
            i += 10
        else if 100 ≤ i < 200 then
            i += 20
        else if 200 ≤ i < 300 then
            i += 50
        else
            i += 10^⌊log10(i)⌋
        end if
    end while
    return windows
end procedure

procedure COMPUTE-MOV-AVG-SCORE(x̄, range, labels)
    best_lower, best_upper = each is list of num_features NaN
    best_score, best_window = −1, −1
    for w in get-windows(range) do
        mov_avg_series = movmean(x̄, w)
        score, lower, upper = compute-constant-score(mov_avg_series, labels)
        if best_score == −1 or score > best_score then
            best_lower, best_upper = lower, upper
            best_score, best_window = score, w
        end if
        if best_score == 1 then
            break
        end if
    end for
    return best_score, best_lower, best_upper, best_window
end procedure
```

lies in datasets whose score is low and vice versa (complex approaches which cannot detect anomalies in datasets whose score is high). The relationship between the simplicity of datasets and the complexity of methods is subtle to grasp. Datasets complexity is driven by several factors, the first of which *should* be temporal dependence. However, if datasets have mainly anomalies separable from normalities in space, the encoding and using the

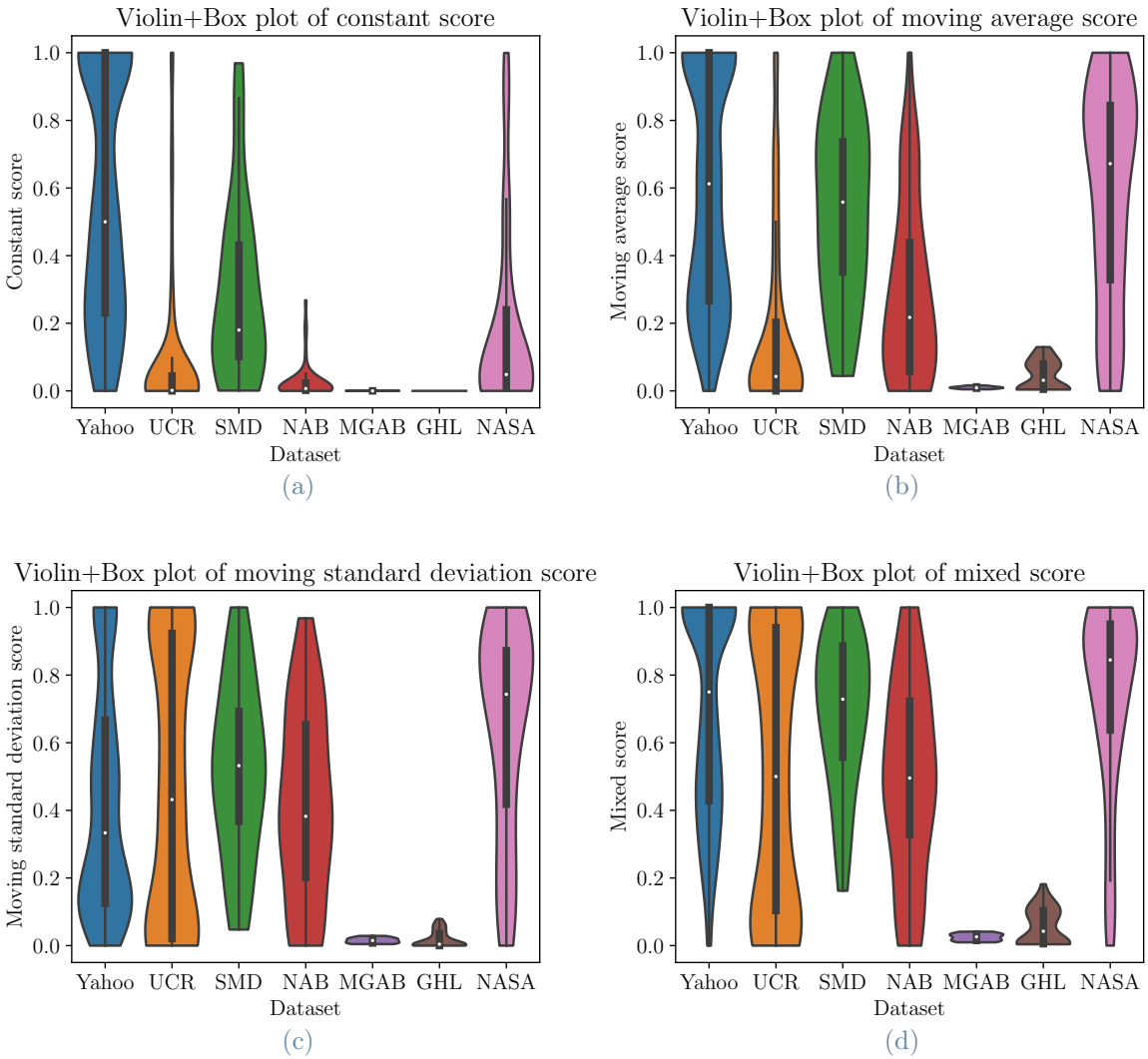


Figure 3.5: Figures of simplicity scores of publicly available benchmarks (one score for each dataset in the benchmark). Inside the violin plot there is a box plot such that the plot shows the distribution of scores and the interquartile range of datasets. The images can be generated by running the script "example_public_ds_simplicity.py". The scores have been computed using as range $[2, 300]$ on the original series till the third order differentiation.

temporal dependency between points may only add complexity (without improving the performance). Figure 3.6 shows examples of simple and complex time series. As [85] observes, it is possible to use simple approaches to find all or most anomalies for many simple datasets. The analysis presented in this chapter aims to provide additional instruments to evaluate the usefulness of methods and a method enabling qualitative analyses of the complexity of approaches. When most anomalies in a used benchmark are spatially sep-

Dataset	Split defined	N. of time series
Yahoo [48]	NO	367
UCR [85]	YES	250
SMD [73]	YES	28
NAB [4]	NO	58
MGAB [76]	NO	10
GHL [33]	YES	49
NASA [41]	YES	82

Table 3.1: A list of public AD benchmarks with the number of series contained, and the information regarding whether the benchmark defines a training and a testing sets.

arable from normalities, is the method complexity justified? Are simple methods unable to detect anomalies? The answers to these questions are fundamental.

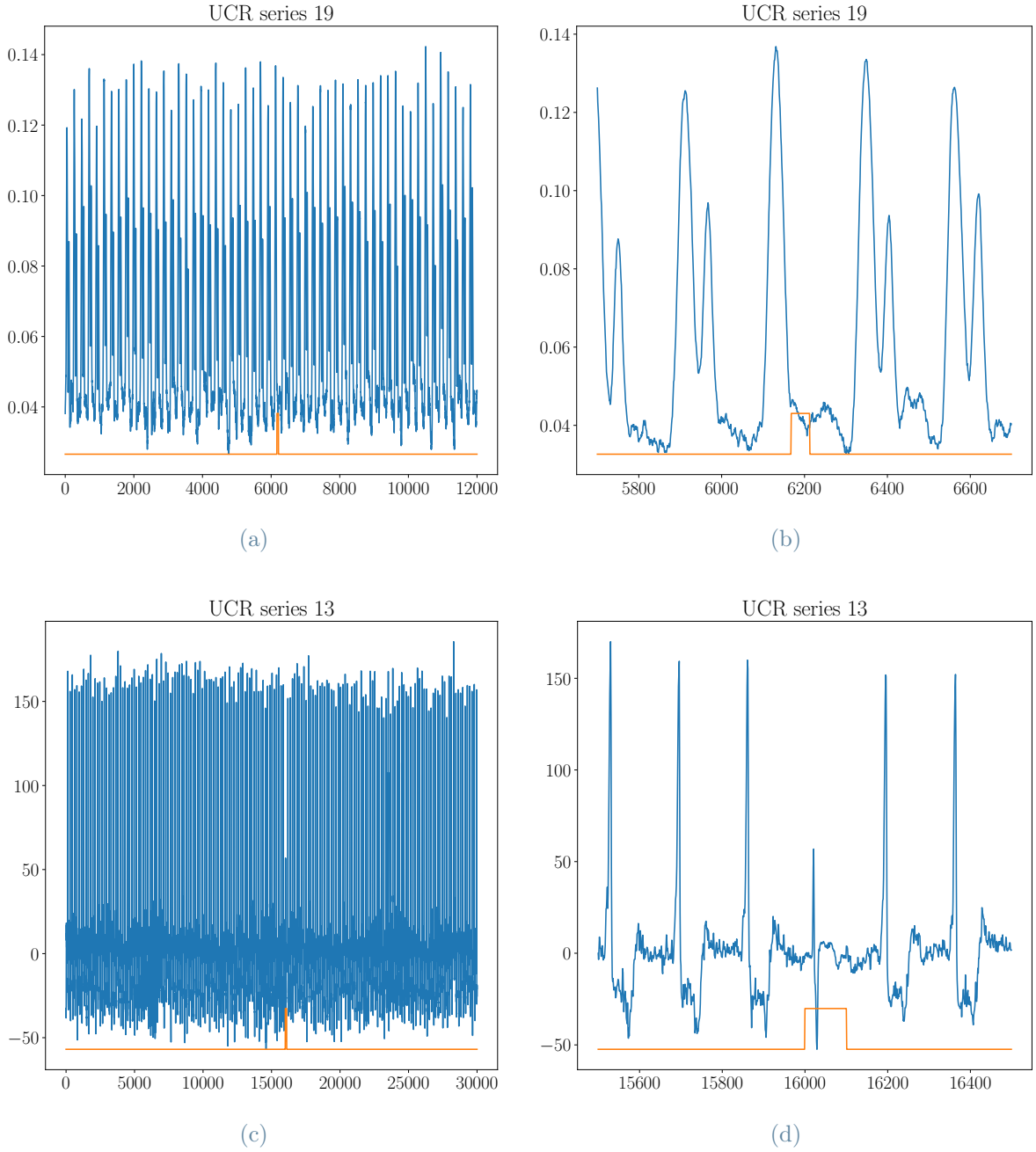


Figure 3.6: The time series in (a, b) is a complex series with a mixed score of 0: no anomaly can be separated in space by normalities. The time series in (c, d) is simple with a moving standard deviation score of 1, while constant and moving average scores are 0. The images can be generated by running the script "example_simple_series.py".

4 | Anomalearn: proposed library for anomaly detection

Creating machine learning models in software is conceptually the same as creating any other piece of software based on mathematics. Software development can follow several approaches, and each has advantages and disadvantages. However, there is a need for a reliable and extensible library for AD highly focused on the development of AD models rather than focusing on the usage of implemented approaches. Since the description of the library involves code, the notation used to distinguish between different objects is detailed in Appendix B.

This chapter describes `anomalearn` (the proposed library). Section 4.1 describes its purpose. Section 4.2 describes the general structure of the library. Section 4.3 describes the approaches for OOP libraries for machine learning. Section 4.4 describes the standard format used by the readers of datasets. Section 4.5 describes the central functionality of the `applications` package. Section 4.6 describes the approaches to develop pipelines and the one used for `anomalearn`. Section 4.7 describes the modular approach used by the library and its extensibility. Section 4.8 describes the implementation of efficient algorithms in Python for `anomalearn`. Section 4.9 describes how to implement state-of-the-art methods using `anomalearn`.

4.1. Purpose of the library

The creation of machine learning models and programs is a complex task. It involves several sub-problems, such as function optimization, handling of high-dimensional data, transformations, and others. Overall, there exist several good libraries implementing famous machine learning approaches [65], implementing statistical learning methods [68], implementing building blocks to create neural networks [2, 64], implementing fast array operations [39], implementing optimization and mathematical procedures [81], implementing data reading operations [63], and many others. However, there are few libraries for AD and the development of new models. Currently, libraries implement some AD methods:

PyOD [89], Orion [7] and Anomalib (for images) [6] are notable examples of AD libraries. PyOD is a collection of anomaly detection algorithms mainly for tabular data. Anomalib is a collection of deep learning algorithms for anomaly detection in images. Orion is a library for unsupervised time series anomaly detection. These libraries are more focused on implementing state-of-the-art models rather than creating a structured approach to develop new models from data reading to output generation. Moreover, many libraries do not have data readers returning data frames or similar common structures for usage. `Anomalearn` aims to create an end-to-end library using a modular approach enabling the independent usage of the library's components: dataset reading objects, pre-processing components, post-processing components, scoring functions, models, and pipelines. The design devotes attention to the easiness of the APIs for simple extendibility and automatic (or quasi-automatic) functioning of new components with the rest of the library.

4.2. General structure of the library

`Anomalearn` contains several objects offering different functionalities. It is possible to distinguish between two classes of packages: core packages and utility packages. The first class of packages offer the most important functionalities of the library and is not used or is slightly used by other packages. The latter class of packages offer either additional functionalities or functionalities only needed for the creation of other packages. The library has four core packages:

- Algorithms: it contains all the algorithms for handling data: processors, transformers, models, tuners, and pipelines. It is structured in sub-packages which are loosely coupled to enable the usage of single modules without the need to know in any way the functionality of other packages or with the need to know only what is strictly necessary, e.g., pipelines make use of other layers to create a chain of operations, it is needed to know the functioning of pipeline elements to understand the pipeline.
- Data analysis: it is a set of functions for analysing data. The essential functions of this package are the dataset simplicity scoring functions. They implement the pseudo-codes of chapter 3 and add an option called "diff" stating that the score will be computed on the series and its differentiations up to the order specified in that field, among the scores, the maximum will be returned.
- Applications: it is a set of functionalities to aid the process of creating experiments. This package contains the code to build experiments or automate the learning process. It is a package based on other packages for automation and reproducibility.

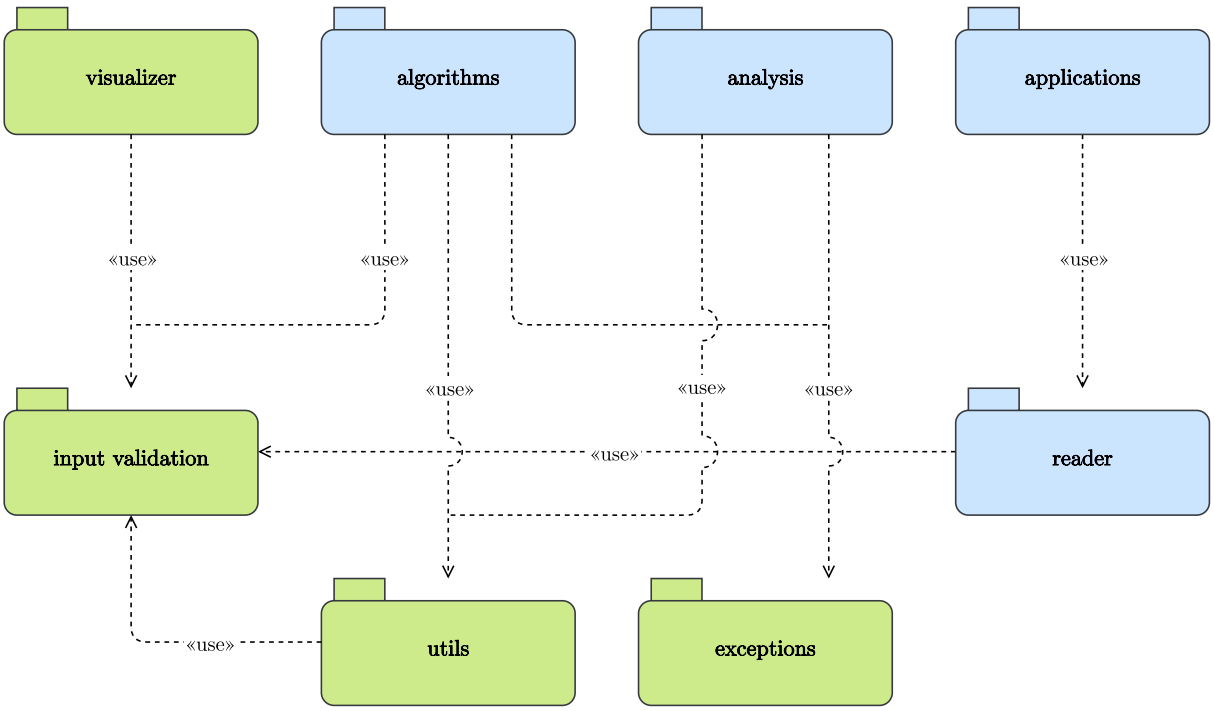


Figure 4.1: UML package diagram of the top level packages of `anomalearn`. Blue packages are the core packages, green packages are utility packages.

- Readers: it is a set of generic and specific readers for general time series and anomaly detection time series datasets. All the readers translate the original format of the dataset into a `pandas` data frame such that the user does not need to know the specific format in which the dataset has been saved.

Figure 4.1 contains the UML package diagram of the top-level packages. It can be noted that there is a low coupling of the core packages: they have few or no incoming usage relationship arrows. Furthermore, almost all the packages do not have sub-packages; therefore, the internal structure of each package is self-elucidative once one opens its folder since they mostly contain interfaces and some implementations. Differently, the structure of the `algorithms` package (figure 4.2) demands a detailed description. Its sub-packages are loosely coupled, `pipelines` and `preprocessing` packages the only ones used by the others. Other packages use the interfaces defined in `pipelines` to implement objects which can be inserted into them. These interfaces are used by concrete pipelines to be able to define a pipeline composed of any sequence of objects that can be inserted as a pipeline layer. Differently, some types of postprocessing techniques depend on the preprocessing technique that is being used before running the model. They use the preprocessing techniques to accomplish their task. The contents of the sub-packages of `algorithm` are:

- `transformers`: it contains objects applying transformations on data.

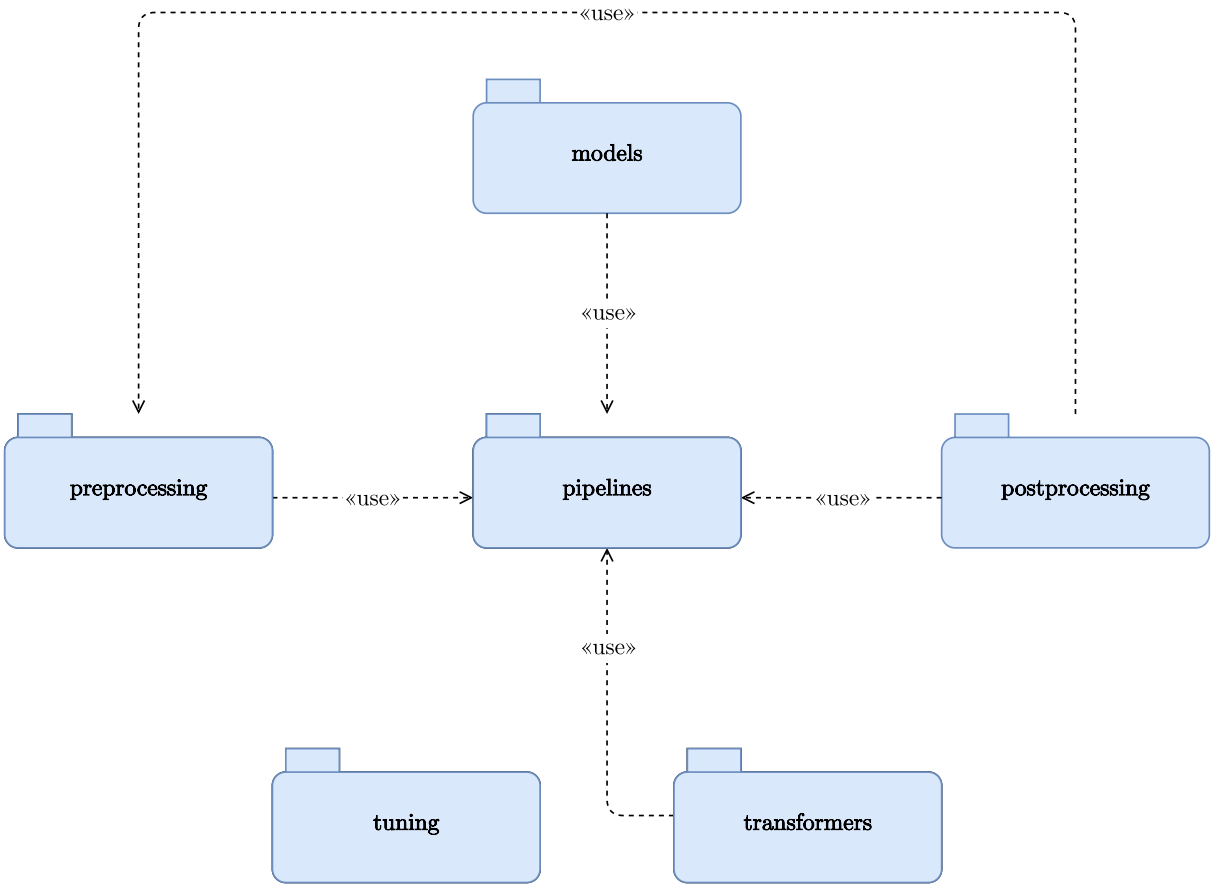


Figure 4.2: UML package diagram of the `algorithm` package.

- `preprocessing`: it contains objects processing data before feeding them into the model.
- `models`: it contains models for computing scores or labels from data.
- `postprocessing`: it contains objects processing the output data of a model.
- `pipelines`: it contains objects defining a way to pipeline all objects that could be inserted as a pipeline layer (typically, objects from the previous packages).
- `tuning`: it contains objects for tuning parameters or hyper-parameters.

4.3. OOP approaches for ML libraries

Most of the machine learning Python libraries follow an OOP approach. In general, each model is implemented as an object able to fit on data (if the model can be fitted) and to predict (classify, cluster, or other). Since libraries adopt several different paradigms, the aim is to mix them to obtain the most usable objects by exploiting all the advantages of

each utilized approach. Before directly describing the proposed library, I will review and introduce two programming styles used by other libraries: duck typing and MLPrimitives.

4.3.1. The duck typing approach

Duck typing is an approach strongly related to dynamic typing. Dynamic typing refers to the act of verifying the type of an object at run-time rather than at compile-time. Python is a dynamically typed programming language since it assigns and checks the type of an object at run-time; therefore, it supports duck typing. The easiest way to introduce it is to quote Python's documentation [1]:

> *«If it looks like a duck and quacks like a duck, it must be a duck.»* — Glossary of Python

It means that the emphasis is on the methods or attributes of an object more than on its type. If an object X takes as input an object of class Y that implements a list Z of methods, every object implementing Z will be accepted since it implements the same methods, even if they have different types. Libraries like `scikit-learn` [16] follow the duck typing approach and offer some base classes already implementing some methods. Besides its simplicity in implementation (one does not have to know any interface, only the naming convention used for methods needs to be known), this approach requires one to know beforehand all the naming rules used for an object. Implementing a new object requires following all the conventions specified in the developer guides detailing all classes of objects. Respecting these rules means that the names of methods and fields must be coherent and consistent with the rest of the API. If an argument of a function is an object implementing a list of methods, the callee must verify whether the object is appropriate or not. So, working with duck typing implies the choice between the following strategies:

- Try to call the required method; if it does not exist, Python will raise an exception.
- Adopt EAFP programming [1].

Besides the choice of how to deal with it, there is an observation about it: it relies on the developer knowing the conventions and reading the documentation. Since most developers tend not to read documentation or devote little attention to it, I think it is avoidable to use such an approach. Enforcing the coherence and consistency between objects by enforcing types should be better. It happens so frequently that developers do not read the documentation that the They Ain't Gonna Read It (TAGRI) principle [8] has been defined in agile modelling.

4.3.2. A different view: MLPrimitives/MLBlocks

A different approach to developing machine learning models is that of MLPrimitives/MLBlocks [69, 80]. They base the library on the concept of primitive:

> *«A primitive is a data processing block. Along with a code that does the processing on the data, a primitive also has an associated JSON file that has a number of annotations.»* — MLPrimitives' documentation

Besides this definition of two sentences, they describe in detail the concept of primitive. The most important property of a primitive is that it can be of two types: function and class. A **function primitive** is a simple function which can be called directly. A **class primitive** is a primitive that must be instantiated before it can be used for computation. For class primitives, there is a second classification: directly importable against primitives requiring an adapter. The library follows a typical pattern in machine learning software packages, the fit-produce abstraction: an estimator has a fitting method for calculating the parameters of the model and a production method for obtaining the output after the estimator has been fitted. An example of such approaches is the `scikit-learn` estimators. Differently, `tensorflow keras` models require compilation before they can be fitted on data. They are an example of a model requiring an adapter to be interfaced with MLPrimitives/MLBlocks. Besides this summary of the functioning of this library, it is vital to describe how a component (which will be a primitive) can be integrated into this ecosystem. Each primitive has an annotation file written in JSON specifying the name of the fitting function and the production function, along with all the hyperparameters. Each primitive can be used only if it has a JSON file with a directly importable field of the primitive implementation, e.g., `sklearn.preprocessing.MinMaxScaler`. Therefore, extending the list of primitives with a new machine learning block requires two pieces: the class or function containing the implementation and the JSON annotation file for that primitive. A positive side of this approach is the freedom in the implementation of new techniques, e.g., the fitting and production functions can have any name since they are specified in the JSON. A class can have any number of methods and is not bound to almost any naming convention. A negative side is that the implementation of a primitive requires the knowledge of the JSON format and the specification of every detail of the function in such a format. Another advantage is the simplicity of embedding any machine learning model following the fit-produce abstraction by only creating the annotation file. However, this approach is not flexible and hardly leverages the automatic refactoring tools offered by many software packages. If the methods of the class under development change the signature, the annotation JSON file must be modified manually.

4.3.3. Proposed approach: interface-driven

The proposed approach for the library is an interface-driven approach aiming at taking the best of both practices and providing tools to ease the development while avoiding overly constraining the user. The idea is to characterize all the possible types of models through interfaces with specific methods. Instead of having one unique production method called `predict`, the library defines several interfaces for machine learning objects like IClassifier or IParametric implementing the methods `classify` and `fit` respectively (there are many other interfaces). Therefore, the machine learning engineer can choose the most appropriate interface (or interfaces) for the model under development while being compliant with the API such that the method can be easily integrated with other library objects. This approach does not provide the same level of freedom of MLPrimitives/MLBlocks while it still provides a higher level of freedom with respect to only having `predict` method for production. At the same time, it also enables the adoption of a duck typing approach if the machine learning engineer feels more comfortable using it, even if it is not recommended to adopt such a practice. The use of duck typing with interfaces is permitted since Python is a dynamically typed programming language offering a method called `__subclasshook__` for abstract classes to enable the customization of the subclass check through `isinstance` function. Interfaces override this method and define an object as a subclass if it implements all the interface methods, e.g. if the interface defines only one method X, any object with a method called X is considered a subclass of this interface. Such an approach significantly simplifies the documentation: if an object of a specific interface is required as input to a method, the documentation can effortlessly state that the required object is expected to expose the same interface methods X. Then, it will be a choice of the user whether to directly inherit from it or manually add the methods and copy the signature of the interface. Neither the former nor the latter way of creating the class is forced. Secondly, once a class has been implemented and tested, nothing else is needed to let it work with other library objects.

Besides this general approach, it is crucial to assume a notation similar to famous libraries to simplify the learning process of `anomalearn`. The methods having the same meaning as methods defined in `scikit-learn` API [16] have the same signature in interfaces, e.g., the `fit` method takes `x` and `y` with optionally other parameters. Thus, switching from a `scikit-learn` estimator to an `anomalearn` estimator should be straightforward. However, the saving mechanism is an essential feature of models and objects of `anomalearn`. A serializable object must implement a safe saving method. Therefore, the use of `pickle` can be avoided to save them since it is unsafe (the only case in which `pickle` is allowed is on adapters of objects that must be saved using `pickle`).

4.4. Standard format for dataset reading

Machine learning tasks cannot be executed without a dataset, and each dataset must be read before being used (the words dataset and benchmark will be used interchangeably to identify a set of time series for AD). Many state-of-the-art algorithms repositories cannot leverage pre-existing dataset readers and require to implement them. This approach is wrong from the point of view of the scientific community: it violates the Don't Repeat Yourself (DRY) principle [42].

> *«Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.»* — Andrew Hunt and David Thomas

Even though this principle refers to the development of a system by a developing team, it can be generalized to the scientific community as a whole. Scientific progress conveys the discovery of new features and the proposal of new approaches. Once a method or a dataset has been proposed, there should be a single implementation of the same procedure for several reasons:

- Errors: every time software is being developed, some errors might appear. With a single piece of software, it will be possible to discover and fix it for every community member. If everyone has his/her implementation, it might be that the implementation has some conceptual errors which won't be found; therefore, it might create confusion in results.
- Coherence: multiple pieces of software doing the same thing might adopt a different format for data or models. With a single publicly accessible element, the scientific community can compare research works more easily.
- Slowed progress: if models or data readers have to be re-implemented every time a new study needs to be carried out, an immeasurable quantity of time will be wasted in re-implementing currently existing functionalities. Differently, if the data readers and models are publicly usable, each scientist can devote all attention to his/her study to bring innovation to the field.

For these reasons, the `anomalearn` library implements several data readers of commonly used datasets in the context of anomaly detection. The data readers expose the same interfaces, and the output format is standard; therefore, the scientist does not need to know the details of the format in which the dataset is saved, and he/she can use different data readers and different datasets using the same methods and fields.

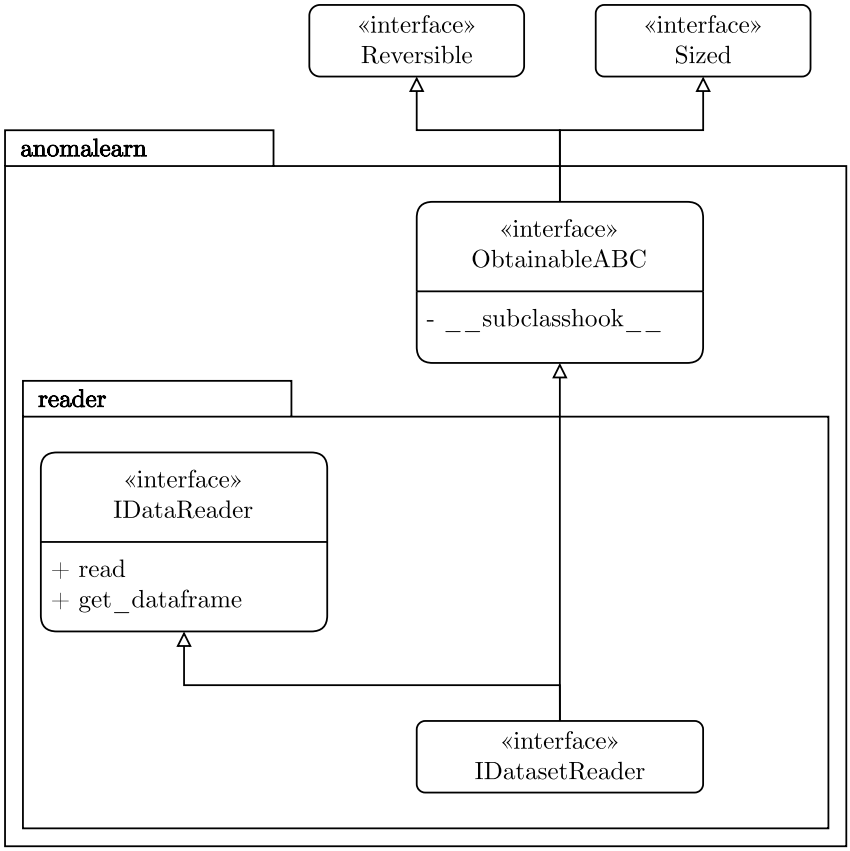


Figure 4.3: Partial UML package diagram of `reader` package. It contains only the necessary elements to understand the IDatasetReader interface.

Firstly, I will describe the format of the datasets:

- **timestamp**: it is the column containing the index.
- **is_training**: it is the binary column identifying the training points. If a point has a 1, it is a training point. *Note that not all datasets have it* (because not all datasets have a pre-defined train-test split).
- **class**: it is the binary column specifying whether a point is an anomaly (1) or a normality (0).
- **value**: it is the column containing the values of a univariate time series. *Note that it is present only for univariate time series.*
- **channel_X**: it is a list of columns in which X is generally a number identifying the channels of the multivariate time series, e.g., a multivariate time series with seven channels will have seven columns starting with **channel_**. *Note that it is present only for multivariate time series.*

```
from matplotlib import pyplot as plt, gridspec

from anomalearn.reader.time_series import SMDReader
from anomalearn.visualizer import line_plot

reader = SMDReader("../data/anomaly_detection/smd")

# iterate over the SMD dataset and plot
for ds in reader:
    fig = plt.figure(figsize=(8, 8), tight_layout=True)
    gs = gridspec.GridSpec(2, 1)

    series = fig.add_subplot(gs[0, 0])
    line_plot(ds["timestamp"].values,
              ds["channel_0"].values,
              ax=series)

    targets = fig.add_subplot(gs[1, 0])
    line_plot(ds["timestamp"].values,
              ds["class"].values,
              ax=targets)

    plt.show()

# read specific time series by an index and print its dataframe
df = reader[0]
print(df)

# read specific time series by name and print its dataframe
df = reader.read("machine-1-1").get_dataframe()
print(df)
```

Listing 4.1: Example of code to read the SMD Dataset

The names of the columns are specified in an INI file called "time_series_config.ini". They are retrieved by the `reader.time_series` package and stored on a public variable called "rts_config" (rts stands for reader time series). Even though this configuration file can be modified to change the names to whatever the user wants, I suggest not doing that. The advantage of a standard naming convention is that results are easily comparable with each other.

Finally, the dataset readers will be described. They are concrete classes implementing the interface IDatasetReader (the class UML diagram in figure 4.3 details it). They are iterators over the series of the dataset whose length is the number of time series. Each time series can be accessed through the indexing operator, and a `pandas` data frame will be returned with the specified fields of the previous list. Moreover, each benchmark is characterized by different types of series, and some datasets also give names to them;

```
from anomalearn.applications import ExperimentLoader
from anomalearn.reader.time_series import YahooS5Reader, MGABReader

reader_yahoo = YahooS5Reader("path_to_yahoo")
reader_mgab = MGABReader("path_to_mgab")

experiment = ExperimentLoader([reader_yahoo, reader_mgab],
                              [(0.8, 0.2), None], # splits
                              [None, [0, 1, 2]], # series to use
                              (0.7, 0.3)) # default split

for train, test in experiment.series_iterator():
    # do stuff
```

Listing 4.2: Example of code to create an experiment over two datasets specified some splits and series to be used.

therefore, the parameter "path" of the "read" function will also accept a name in some cases. In such a way, the user will be able to iterate over all the series and pick any of the series either using an index or the name of the series. A snippet of code providing an example of such capabilities is presented in listing 4.1. The snippet uses the SMD as an example since its time series have names, i.e., all three behaviours can be shown.

4.5. Experiments

Research in the machine learning community usually involves the usage of a dataset to evaluate the capabilities of a method or to make a proof of concept. Moreover, testing and evaluating models on various benchmarks is common since they have different characteristics. Splitting data, creating cross-validation sets for evaluation, and many other functionalities are available in machine learning libraries such as `scikit-learn` [65]. However, also the creation of a set of experiments involving training and testing on several datasets is an highly repetitive task that one would like to automate. Since there are no data readers available in AD libraries, there is no general object to automate this process. Given benchmarks, a researcher may want to test his/her model on the overall dataset or on a subset of the series contained in it, as well as he/she may want to define different train-test splits for distinct datasets. Given the existence of the data readers in the library, it is possible to implement an experiment object capable of retrieving and splitting time series from benchmarks in any order specified by the user. These objects are located in the `applications` package, and they use the interfaces defined in the `reader` package to operate such that any object exposing this interface can be used, also if it is not part of the library (see figure 4.1 use relationship). The ExperimentLoader is the central

component of the `applications` package. It is a collection of readers, with optionally specified splits and series to be used (`None` means "use the default value"). It will iterate on the selected series for each dataset (if nothing is specified, all the series are retrieved) and divide the dataset with the specified split, if any; otherwise, it will use the default one. To ease the creation of experiments, an iterator over the series is also implemented such that a simple and intuitive usage as in listing 4.2 can be adopted to get the data frames of the time series of the selected AD benchmarks.

4.6. Pipelines: avoid repeating code

The task of subsequently applying transformations to data before the actual training of the model is as repetitive as splitting and selecting the training set. These transformations are called preprocessing operations since they happen before data processing by the model. Contrariwise, transformations performed on the output of the model are called postprocessing operations, and they are frequent too. The former type of transformation might be necessary, e.g., there are categorical features, and the model only accepts numerical input. If this is the case, several transformations can be used to transform data in an acceptable form for the model, e.g., one-hot encoding is an example of such a transformation: it changes the dimension of the input data by increasing them by a factor of $n - 1$, where n is the number of categories of the encoded feature. However, transformations may not change the dimensions of the input like range transformations, e.g., min-max scaling. The usage of scaling transformations can be of interest both before and after the model, e.g., scaling the output of the model in range $[0, 1]$ when the model outputs abnormality scores enables to represent an anomaly with a value of 1 and normal points with the value 0. Furthermore, the sliding window preprocessing operation is important when a spatial model is used to perform anomaly detection: temporal data must be modified to become spatial data as well as the output of the model must be inversely projected to a temporal dimension. Given the repetitiveness of such operations, libraries introduced the concept of pipeline: a compound object defining how the contained objects interact to transform the input into an output. The simplest example of a pipeline is the sequential pipeline: a pipeline composed of a list of objects which must be applied sequentially, i.e., the output of the i^{th} object is the input of the $(i + 1)^{th}$ object.

4.6.1. The scikit-learn pipeline

The `scikit-learn` pipeline is the first pipeline object that should be described because it is contained in one of the most commonly used libraries. It has two main types of

```
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.svm import LinearSVC

rng = np.random.default_rng(seed=123)
data = rng.random((100, 3))
labels = rng.integers(0, 2, size=100)

pipe = Pipeline([("std", StandardScaler()),
                 ("minmax", MinMaxScaler()),
                 ("linear_svc", LinearSVC())])
out = pipe.fit(data, labels).predict(data)
```

Listing 4.3: Example of `scikit-learn` pipeline with estimator at the end.

pipelines: Pipeline and FeatureUnion. The former is a somewhat simple concept. It consists of a sequence of transformations and an estimator at the end. The latter is constituted of a list of transformer objects: the transformers are applied in parallel, and their output is horizontally stacked. This means that the shape of the input will change from `(n_samples, n_features)` to `(n_samples, n_components)` where `n_components` is the sum of the number of features in output to each transformer.

Listing 4.3 shows an example of `scikit-learn` pipeline with an estimator at the end. It is possible to observe that this type of pipeline is what I previously called a sequential pipeline. However, it is slightly less general than a sequence of operations. It only accepts one estimator object at the end. It means that it is impossible to create a pipeline for meta-learning (learning from the output of a learner instead of learning directly from data) since there must be only one estimator. Since it assumes that the last element is an estimator, many of the methods of this pipeline (like `predict`) are called on the last element of the pipeline, except for the `transform` method that is called on all the other elements (it is possible to call it also on the last). Therefore, this pipeline focuses on the last element instead of the general compound object. Although it is possible to create a pipeline solely composed of transformations, it is impossible to create a pipeline having either transformers after an estimator or multiple estimators. This is the first notable limitation of the `scikit-learn` pipeline. Contrarily, a nice feature of a `scikit-learn` pipeline is that it can contain other pipelines (if by unwrapping them, the previous constraints are not violated, i.e., only one estimator at the end). Moreover, a change in shape to more than two dimensions is not expected since `scikit-learn` estimators tend to work on 2D data, which means that the usual transformations performed on a time series as preprocessing (e.g., sliding windows) are not naturally defined for this type of pipeline.

Essentially, the preprocessing operations on a time series are more general.

4.6.2. The orion and MLBlocks pipeline

The `orion` pipeline [7] is a wrapper around the `MLBlocks` pipeline [69], and it has been developed for unsupervised time series anomaly detection. The Orion object is in charge of executing the AD functionalities and of the interactions with the underlying `MLBlocks` pipeline. Therefore, the introduction of `MLBlocks` pipeline also describes the pipeline adopted by `orion` (it is a wrapper). Even though this pipeline follows an analogous approach to that of `scikit-learn`, it is much more elaborate. A `MLBlocks` pipeline is a sequence of primitives, no matter which type, it neither assumes the presence of an estimator at the end (even though they report that it is common for a pipeline to have such a behaviour) nor presumes any ordering between primitives. Moreover, the pipeline follows the general approach of the library: it is attached to an annotation file as any other primitive. Furthermore, it wraps primitives in MLBlock objects: a container with the name of the primitive and a counter. This tuple of two elements (the name of the primitive and a counter) is sufficient to fully describe a block because the library has a detailed lookup procedure to find also the user-defined primitives (see "Adding Primitives" part of the `MLBlocks` documentation for more details). However, since the objects are not directly passed to the pipeline as in the `scikit-learn` pipeline, if any of its layers requires some initialization parameters different from the default ones, they must be passed to the pipeline. Upon the instantiation of the MLPipeline, it is possible to specify both the list of primitives contained in it and a dictionary of initialization parameters in which the keys are the primitives of the pipeline. In essence, the `MLBlocks` pipeline is created following a descriptive approach rather than an imperative approach: it receives a list of names of primitives and a dictionary of instantiation parameters, then it creates and instantiates each object of the pipeline.

Even though the pipeline is sequential, input/output management between layers is comprehensive. It uses a structure called **context dictionary** to keep track of each variable. Before explaining its functioning, recall that each primitive has an associated annotation file in which the fit and produce functions have a list of arguments with names and types; additionally, the production function also has a list of the outputs with names and types. The context dictionary maps the names of variables to their content. Each time a layer has to be executed, all the variables in the context dictionary with the same name of the arguments accepted by the layer are passed to it. Once the output is generated, there are three possible scenarios:

- Overwrite: all the output variables have a name present in the context dictionary. In this case, the variables in the context dictionary with these names will be overwritten.
- Partial overwrite: some of the output variables of the layer have the same name as some variables in the context dictionary. Variables whose name is present in the context dictionary will overwrite the current ones. The other variables will be added to the context dictionary.
- Addition: all the output variables have names which are not keys of the context dictionary. All the variables will be added to it.

This approach has a significant advantage: each layer can add and overwrite variables in a shared context dictionary. Therefore, it is possible to write heuristic functions computing the parameters to pass to the subsequent layers as well as any manipulation function.

4.6.3. The anomalearn pipeline

The `anomalearn` pipeline tries to mix both approaches and follows the interface-based policy presented in section 4.3.3. The approach of `scikit-learn` includes several constraints to the types of pipelines that can be created; the approach of `MLBlocks` gives complete freedom of development unless the annotation JSON file is provided. The policy of `anomalearn` tries to create a balance between the two: it does not need any annotation procedure while giving a lot of freedom in the development. First of all, the pipelines of `anomalearn` have to satisfy the following properties:

- Savable: each pipeline must be savable. It does not matter if the layers of the pipeline include a save method or not. The pipeline must be safely serializable to enable loading and publishing. The reason is that when a sequence of layers has been trained to carry on a specific task, the developer should be allowed to store and distribute his/her work easily.
- Interface-based: each pipeline must implement the same interface. A pipeline represents a concept: a compound object with explicitly stated connections between layers. The way in which the layers are connected is dependent on the implementation.
- A pipeline is a layer: a pipeline must be a valid layer for another pipeline, independently of its content.
- A layer is a concept: a layer is an object implementing a given interface. A pipeline

must accept as a layer any object implementing the interface or implementing the methods defined in the interface.

These requirements make the design of a pipeline complex, but it also makes the pipeline simple and easy to use. Particularly, the fourth element of the list is the reason why the `pipelines` package is used by almost all other sub-packages of `algorithm` (see figure 4.2). The elements which can be included in a pipeline as a layer implement one of the abstract classes for pipeline layers. There are two abstract classes: an abstract class for layers which does not implement a saving method and an abstract class for layers that implement a saving method. In this way, the pipeline knows whether an object must be re-instantiated while loading from a file or loaded using the loading method of its class. The loading or instantiation operation requires knowing the type of object. Each object deriving from the class SavableModel of `anomalearn` creates a file called "signature.json" upon saving. This JSON file contains the name of the class that is being saved. Therefore, the pipeline only needs to open the signature file, and it will know which class to load. Conversely, unsavable objects need to be instantiated instead of loaded. However, since the pipeline must be decoupled from the other objects, it uses two utility functions of the `anomalearn.algorithms.algo_functions` module to load classes: "load_estimator" and "instantiate_estimator". These functions can load and instantiate a model of the `anomalearn` library or a list of user-defined classes. They only need the name of the class to be loaded or to be instantiated. They first check from the list of user-defined classes; if the class is not contained in the list, it looks up in the folders of the library for classes with that name for instantiation or loading. If the class does not exist, an exception is raised.

Besides the description of the saving mechanism, the APIs can be described quite easily. The Pipeline accepts only one argument at creation: a list of layers. The list of layers is a list of several types of input: a tuple of different dimensions or a layer object. Each layer in a Pipeline has a unique name for identification purposes and a boolean value stating if it has to be trained during the fit. The only required argument for an element of the list is the layer object: the name defaults to the string representation of the object and a number, and the boolean defaults to true (the trainable flag does not state that the pipeline will try to execute the fit method. It state that the layer will be trained if it has a fitting method). The elements of the list can be an object or a tuple whose order of its content is: name, object, and boolean. The input list can contain one, two or three elements; the order (name, object, boolean) and the presence of the layer object are the only requirements. A Pipeline can be empty and is a mutable object: layers can be added, removed and modified. The listing 4.4 contains a snippet of code with an example

```
from anomalearn.algorithms.models.machine_learning import
IsolationForest
from anomalearn.algorithms.pipelines import Pipeline
from anomalearn.algorithms.transformers import MinMaxScaler

minmax = MinMaxScaler()
isolation_forest = IsolationForest()

pipeline = Pipeline([("minmax", minmax),
                     (isolation_forest, True)])
pipeline.summary()
```

Listing 4.4: Example of `anomalearn` pipeline with a scaler and a model at the end.

of Pipeline's creation.

4.7. Modularity and extensibility

The library is composed of packages (see figures 4.1 and 4.2) with low coupling. In this section, the word module refers mainly to functionalities rather than strict Python modules (even if the two coincide most of the times). To boost the extensibility and the simplicity of the implementation of new approaches, the philosophy behind the development of the library can be stated in the following sentence:

> *If a functionality has to be added to the library, it should also be removable without hindering the usage of the library.*

It means that a functionality should be the least dependent on other modules. Ideally, the addition of a new functionality does not require the modification of other pre-existing functionalities in the library. In listing 4.4, the component MinMaxScaler is used, it can be removed without causing problems to the rest of the library at this state. This behaviour should be achieved from each new component introduced in core packages. This practice will be called the Add If Removable And Independent (AIRAI) principle. It states that a functionality can be integrated if it can also be deleted immediately after it has been added and if any other functionality of the core packages can be deleted too once the new component is integrated. It does not state that the functionality must be removable for the whole life cycle of the library. Example: a state-of-the-art approach is implemented using the `anomalearn` pipeline and added to the library. It is acceptable that the deletion of layers used by this approach will cause dependency problems.

Moreover, the implementation of any approach should also respect the conceptual division of packages. Example of wrong implementation: if a model is added to the models package

and it uses a preprocessing object, or it does an equivalent job to that of a preprocessing, it should be a pipeline. Specifically, the core implementation after the preprocessing can be implemented in models. The overall architecture should be a pipeline. However, there exist exceptions to this principle: postprocessing operations defined only if there was a specific preprocessing before. In this case, the principle applies to the two classes since they jointly represent the new functionality. The principle applies to each introduced functionality at the maximum granularity. Currently, the AIRAI is respected by all objects of the library. Following this principle gives the following advantages:

- Free usage: the use of a single component does not require the knowledge of another component of the library. Any module can be used independently or almost independently of other components (like some postprocessing).
- Implementation simplicity: new functionalities do not need the usage of pre-existing functionalities unless they are pipelines.
- Testing simplicity: testing will be a lot easier. New functionalities can be tested completely by unit testing without the need for integration testing in most cases.

Because of the aforementioned reasons, the low coupling, and the clarity of the interfaces, the system should feel extensible. Each object implemented in the library exposes public methods defined in one or multiple interfaces. Moreover, the arguments of methods are expressed as input interfaces rather than concrete objects, which makes it easier to understand which pool of objects it can work with.

4.8. A note on efficiency

Writing efficient code requires the design of algorithms with the best achievable time complexity. An algorithm with time complexity $\Theta(N^{10})$ won't be efficient in any programming language. However, since Python has been chosen as the programming language of this anomaly detection library, efficiency is an enormous problem. Since it is known that Python is not an efficient programming language, several libraries have been published to write efficient Python code: `Cython` [10, 14], `pythran` [37] and `numba` [47] are notable examples. If methods can be implemented using efficient libraries for array and matrix computation, using Python may be good enough. However, if it is necessary to write an algorithm having loops in Python, it may be the case of using one of the previous libraries. Each library has its advantages and disadvantages. For this note, two properties will be evaluated:

- Simplicity: the degree of simplicity of using the library compared to using Python.

Function	Before numba	After numba
constant score	452.15 s	1.90 s
moving average score	15848.92 s	56.25 s
moving standard deviation score	20529.57 s	63.23 s

Table 4.1: The speed of the simplicity scoring function before and after the usage of `numba`

- Compilation: whether the library requires compilation or not; thus, whether the library allows the development of pure Python packages.

`Cython` and `pythran` both require compilation of Python code to use the efficient version, whereas `numba` employs a JIT compiler based on decorators of functions and classes. `Cython` is a super set of the Python programming language more than a compiler. It allows to invoke C functions directly and adds new constructs directly mapping to C code to create efficient code. Differently, `pythran` and `numba` are not super sets of the Python programming language. They add decorations to code to produce efficient code. The former adds these "decorations" as comments with only one command (pythran export). The latter introduces some decorators of functions and classes that state how to compile the Python code. However, `pythran` requires compilation of the Python file to produce efficient code, while `numba` will compile the function the first time it is called, and this compilation will last for the whole duration of the session. Given this short introduction of these libraries, `numba` has been chosen to produce efficient Python code because of its simplicity and capability of producing pure Python packages for distribution. Table 4.1 contains a list of the speed of simplicity scores' algorithms before and after using `numba`. As can be seen, it provides huge improvements. Moreover, it also allows leaving the code almost identical to the Python version.

4.9. Reproduction of state-of-the-art methods

State-of-the-art approaches can be implemented in `anomalearn`. However, it is important to keep attention to the components of such approaches. Many state-of-the-art approaches include preprocessing or postprocessing operations other than a model. A typical preprocessing procedure in time series AD is the computation of the sliding windows. Then, it is also common for reconstruction (or forecasting) approaches to compute the score of a point based on the median/mean reconstructed (or forecasted) value or to compute the score based on a gaussian distribution over the vector of all reconstructions (or forecasts). The thresholding method to extract labels from scores is instead a postprocessing operation. In these cases, the state-of-the-art approach presented in an academic paper is the

compound object of preprocessing operations, model, and postprocessing operations in which the model solves a task different from AD (clearly, forecasting is not a classification task). Therefore, it is common for state-of-the-art approaches to be pipelines.

5 | Conclusions

This thesis presents a series of algorithms to evaluate the simplicity of AD datasets and a library for the development of AD methods on time series. The work on the analysis of simplicity provides a tool for assessing the practical complexity of a dataset, a task that is becoming increasingly important. Conversely, the need for data to perform experiments and evaluate models is less prominent (there are several publicly available datasets). However, given the rise of new datasets, a new challenge reveals itself to the scientific community: the evaluation of the properties of a dataset. The importance of using various and different datasets is vital for the development of knowledge and methods. The proposed definitions of simplicity and algorithms give a consistent, reproducible and helpful tool to evaluate datasets. They establish a formal definition of a score to assess the simplicity of a model based on statistical measures. These definitions are independent of the algorithms, allowing different implementations to compute them. Furthermore, the proposed algorithms calculate such a score with a time complexity of $\Theta(NF\log(N) + ANF)$. These algorithms automate the process of evaluating the level of simplicity of the datasets and are sufficient conditions: if a dataset has a high simplicity score, it is simple. Contrariwise, a low score does not mean that the dataset is so complex that few methods are able to deal with it. They aid data scientists in determining the simplicity of datasets, such that they can test their model on both complex and trivial time series and such that a more comprehensive evaluation of the proposed approach can be carried out. These tools equip the data scientists with a method enabling the choice of the datasets for evaluating the model in identifying anomalies at varying difficulties. Instead of solely reporting the average performance on a benchmark, it is now possible to record the performance at different levels of complexity of the dataset. Therefore, these tools help the data scientist to answer questions like:

- Is the algorithm capable of solving complex datasets?
- Is the algorithm good at identifying simple anomalies?
- Can the algorithm identify point anomalies?

These tools enable the explanation of the goodness of the method in terms of the com-

plexity of the data that it is able to handle. However, the introduced explainability is also beneficial for creating ensembles of methods. Since the three measures of the complexity of the datasets capture different concepts, the ensembling of models can now be guided by the information on the type of complexity solved by diverse models (however, the test information must not guide the ensembling). It will be possible to ensemble models capable of finding different types of anomalies by observing the kind of complexity of the analysed series, which is a primary aspect of ensembling: models must be distinct and learn diverse patterns/concepts. To wrap it up, evaluating the complexity of algorithms offers both a way to assess the quality of datasets and a tool to ease the creation of ensembles.

The second proposal of this thesis is `anomalearn`. The need for libraries for developing machine and deep learning models for AD is high. The proposed library aids the data scientist in several ways. It does not only help the data scientist in the creation of a new component for AD, it assists the data scientist in creating and evaluating any element of the sequence of operations to be performed on AD. It enables the development of a new preprocessing, postprocessing, or model without re-implementing the other components and the logic to let them work together, e.g., if a researcher invented a new thresholding technique, it is now possible to use all the preprocessing and model components in a pipeline with the new postprocessing operation (the thresholding) to evaluate it against the other pre-existent thresholding mechanisms. Therefore, comparing new and existing techniques is simple, and the data scientist is able to save a massive amount of time that otherwise will be dedicated to implementing existing approaches or the connection logic between components. Furthermore, assessing different preprocessing or postprocessing operations is simple since the pipelines enable the creation of any sequence of operations to train and predict. It allows the creation of compound objects composed of existing models and preprocessing objects to evaluate new postprocessing operations or any different combination of new and existing elements. Moreover, studies about the optimal configuration of methods from preprocessing to output generation can be carried out effortlessly. Therefore, answering the following questions will be substantially simple compared to the current state-of-the-art libraries for AD:

- Is a given preprocessing better in general for these models?
- Is a given postprocessing better with these models?
- Is a given postprocessing better in general?

The approach of the proposed library is general and extensible, which means that adding new functionalities will be easier with respect to other libraries. However, there is still one contribution to the scientific community of this library: the distribution. Currently,

libraries do not implement a safe and simple procedure to save and distribute someone's work. Libraries such as `orion` [7], `MLBlocks` [69] and `scikit-learn` [65] either do not provide a way to save the trained models or do not provide a safe way to serialize the trained models by employing pickle to store objects. On the contrary, `anomalearn` provides a safe way to distribute methods by enforcing the implementation of a saving method that uses secure formats like `npy` files of `numpy` [39]. Compound objects (pipelines) use this function to save the object and its components safely. Therefore, the distribution of trained models with weights is now possible and safe, allowing researchers to distribute their work and make it accessible to the scientific community to boost the development of scientific knowledge and progress. Finally, being the library highly based on interfaces and on a general design using UML, the translation of this library to other languages (e.g., C++) will be much less complex than translating other libraries adopting duck typing or complex approaches which load components at run-time. Therefore, the effort in the design of a simple API does not constrain the library implementation to Python.

6 | Future work

Since the thesis proposed two main contributions, there are two distinct future research directions to describe: simplicity algorithms and `anomalearn`

The simplicity algorithms use statistical measures to evaluate the simplicity of an AD dataset. The description of the complexity of an AD dataset could be enlarged and compared to complexity measures used in other research fields, as well as to other complexity measures for time series [74] which are not devoted to measuring the complexity of AD datasets. Moreover, since the time complexity of the presented algorithms to compute such scores have time complexity $\Theta(NF\log(N) + ANF)$, improving the algorithms to lower the complexity to be linear in the number of points and features $\Theta(NF)$ using related problems such as that of finding the minimum bounding box of a set of points. In the end, a new algorithm to refine the search of the scores or new and better heuristics are possible research directions (currently, since not all windows are tried, it is only an estimation).

Regarding `anomalearn`, the continuous implementation of state-of-the-art and historical approaches in the libraries and the implementation of other preprocessing and postprocessing operations is a valuable future work that will make the library more usable and helpful. Moreover, the implementation of different types of pipeline with parallelism (similar to `scikit-learn` [65] FeatureUnion) is of interest such that multiple transforming operations can be executed in parallel as well as implementing a pipeline able to fit and train estimators in parallel; thus providing objects capable of creating ensembles of the output of several models. Finally, the creation of a structure similar to that of the context dictionary present in `MLBlocks` [69] to handle the creation of named variables to pass to the following layers of the pipeline.

Bibliography

[1] Glossary of Python. URL https://docs.python.org/3/glossary.html#term-duck-typing. Accessed on 2023/03/20.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[3] S. Agrawal and J. Agrawal. Survey on anomaly detection using data mining techniques. *Procedia Computer Science*, 60:708–713, Jan 2015. ISSN 1877-0509. URL https://www.sciencedirect.com/science/article/pii/S1877050915023479.

[4] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, Nov 2017. ISSN 0925-2312. URL https://www.sciencedirect.com/science/article/pii/S0925231217309864.

[5] M. Ahmed, A. Naser Mahmood, and J. Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31, Jan 2016. ISSN 1084-8045. URL https://www.sciencedirect.com/science/article/pii/S1084804515002891.

[6] S. Akcay, D. Ameln, A. Vaidya, B. Lakshmanan, N. Ahuja, and U. Genc. Anomalib: A Deep Learning Library for Anomaly Detection, 2 2022.

[7] S. Alnegheimish, D. Liu, C. Sala, L. Berti-Equille, and K. Veeramachaneni. Sintel: A machine learning framework to extract insights from signals. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22,

page 1855–1865. Association for Computing Machinery, 2022. doi: 10.1145/3514221.3517910.

[8] S. W. Ambler. The TAGRI (They Ain't Gonna Read It) Principle. URL `http://agilemodeling.com/essays/tagri.htm`. Accessed on 2023/03/20.

[9] R. G. Bartle. *The elements of integration and Lebesgue measure*. John Wiley & Sons, Inc., Jan 1995. doi: 10.1002/9781118164471. URL `https://doi.org/10.1002/9781118164471`.

[10] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, and K. Smith. Cython: The Best of Both Worlds. *Computing in Science Engineering*, 13(2):31 –39, 2011. ISSN 1521-9615. doi: 10.1109/MCSE.2010.118.

[11] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, January 2006. URL `https://www.microsoft.com/en-us/research/publication/pattern-recognition-machine-learning/`.

[12] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, NY, 1st edition, 2006. URL `https://link.springer.com/gb/book/9780387310732`.

[13] S. Bittanti. *Model Identification and Data Analysis*. John Wiley & Sons, Ltd, 2019. ISBN 9781119546405. doi: 10.1002/9781119546405. URL `https://doi.org/10.1002/9781119546405`.

[14] R. Bradshaw, S. Behnel, D. Seljebotn, G. Ewing, et al. The cython compiler. URL `http://cython.org`.

[15] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: Identifying density-based local outliers. *SIGMOD Rec.*, 29(2):93–104, may 2000. ISSN 0163-5808. doi: 10.1145/335191.335388. URL `https://doi.org/10.1145/335191.335388`.

[16] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[17] S. Bulusu, B. Kailkhura, B. Li, P. K. Varshney, and D. Song. Anomalous example detection in deep learning: A survey. *IEEE Access*, 8:132330–132347, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.3010274. URL `https://doi.org/10.1109/ACCESS.2020.3010274`.

[18] C. I. Challu, P. Jiang, Y. Nian Wu, and L. Callot. Deep generative model with hierarchical latent factors for time series anomaly detection. In G. Camps-Valls, F. J. R. Ruiz, and I. Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 1643–1654. PMLR, 28–30 Mar 2022. URL https://proceedings.mlr.press/v151/challu22a.html.

[19] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Comput. Surv.*, 41(3), jul 2009. ISSN 0360-0300. doi: 10.1145/1541880.1541882. URL https://doi.org/10.1145/1541880.1541882.

[20] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection for discrete sequences: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 24(5):823–839, 2012. ISSN 1558-2191. doi: 10.1109/TKDE.2010.235. URL https://doi.org/10.1109/TKDE.2010.235.

[21] Z. Chen, D. Chen, X. Zhang, Z. Yuan, and X. Cheng. Learning graph structures with transformer for multivariate time-series anomaly detection in iot. *IEEE Internet of Things Journal*, 9(12):9179–9189, 2022. ISSN 2327-4662. doi: 10.1109/JIOT.2021.3100509. URL https://doi.org/10.1109/JIOT.2021.3100509.

[22] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning. Stl: A seasonal-trend decomposition procedure based on loess. *Journal of Official Statistics*, 6:3–33, Jan 1990.

[23] A. A. Cook, G. Mısırlı, and Z. Fan. Anomaly detection for iot time-series data: A survey. *IEEE Internet of Things Journal*, 7(7):6481–6494, 2020. ISSN 2327-4662. doi: 10.1109/JIOT.2019.2958185. URL https://doi.org/10.1109/JIOT.2019.2958185.

[24] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester. *A Modern Introduction to Probability and Statistics: Understanding why and how*. Springer London, London, 1st edition, 2005. doi: 10.1007/1-84628-168-7. URL https://doi.org/10.1007/1-84628-168-7.

[25] A. Dokumentov and R. J. Hyndman. Str: A seasonal-trend decomposition procedure based on regression. Technical report, Monash University, Department of Econometrics and Business Statistics, June 2015. URL https://www.monash.edu/business/ebs/research/publications/ebs/wp13-15.pdf. Accessed 2023-02-22.

[26] A. Dokumentov and R. J. Hyndman. Str: Seasonal-trend decomposition using regres-

sion. *INFORMS Journal on Data Science*, 1(1):50–62, Apr 2022. ISSN 2694-4022. doi: 10.1287/ijds.2021.0004. URL https://doi.org/10.1287/ijds.2021.0004.

[27] D. Dua and C. Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[28] S. C. Endres, C. Sandrock, and W. W. Focke. A simplicial homology algorithm for lipschitz optimisation. *Journal of Global Optimization*, 72(2):181–217, Oct 2018. ISSN 1573-2916. doi: 10.1007/s10898-018-0645-y. URL https://doi.org/10.1007/s10898-018-0645-y.

[29] W. Feller. *An Introduction to Probability Theory and Its Applications, Volume 1, 3rd Edition*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., 1968.

[30] C. Feng and P. Tian. Time series anomaly detection for cyber-physical systems via neural system identification and bayesian filtering. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 2858–2867, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383325. doi: 10.1145/3447548.3467137. URL https://doi.org/10.1145/3447548.3467137.

[31] W. F. Ferger. The nature and use of the harmonic mean. *Journal of the American Statistical Association*, 26(173):36–40, 1931. doi: 10.1080/01621459.1931.10503148. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1931.10503148.

[32] A. Fernández, S. del Río, N. V. Chawla, and F. Herrera. An insight into imbalanced big data classification: outcomes and challenges. *Complex & Intelligent Systems*, 3(2):105–120, Jun 2017. ISSN 2198-6053. doi: 10.1007/s40747-017-0037-9. URL https://doi.org/10.1007/s40747-017-0037-9.

[33] P. Filonov, A. Lavrentyev, and A. Vorontsov. Multivariate industrial time series with cyber-attack simulation: Fault detection using an lstm-based predictive data model, 2016. URL https://arxiv.org/abs/1612.06676.

[34] A. Geiger, D. Liu, S. Alnegheimish, A. Cuesta-Infante, and K. Veeramachaneni. Tadgan: Time series anomaly detection using generative adversarial networks. In *2020 IEEE International Conference on Big Data (Big Data)*, 2020 IEEE International Conference on Big Data (Big Data), pages 33–43, 2020. doi: 10.1109/BigData50022.2020.9378139. URL https://doi.org/10.1109/BigData50022.2020.9378139.

[35] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. URL http://www.deeplearningbook.org.

[36] M. Gösgens, A. Zhiyanov, A. Tikhonov, and L. Prokhorenkova. Good classification measures and how to find them. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 17136–17147. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/8e489b4966fe8f703b5be647f1cbae63-Paper.pdf.

[37] S. Guelton, P. Brunet, M. Amini, A. Merlini, X. Corbillon, and A. Raynaud. Pythran: Enabling static optimization of scientific python programs. *Computational Science & Discovery*, 8(1):014001, 2015.

[38] A. Gut. *An Intermediate Course in Probability (Second Edition)*. Springer New York, New York, NY, 2nd edition, 2009. doi: 10.1007/978-1-4419-0162-0. URL https://doi.org/10.1007/978-1-4419-0162-0.

[39] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

[40] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[41] K. Hundman, V. Constantinou, C. Laporte, I. Colwell, and T. Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 387–395, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219845. URL https://doi.org/10.1145/3219819.3219845.

[42] A. Hunt and D. Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, Oct 1999. ISBN 020161622X. URL https://www.oreilly.com/library/view/the-pragmatic-programmer/020161622X/.

[43] R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts,

Melbourne, Australia, 3rd edition, 2021. URL https://otexts.com/fpp3/. Accessed Oct. 04, 2022.

[44] V. Jacob, F. Song, A. Stiegler, B. Rad, Y. Diao, and N. Tatbul. Exathlon: A benchmark for explainable anomaly detection over time series. *Proc. VLDB Endow.*, 14(11):2613–2626, oct 2021. ISSN 2150-8097. doi: 10.14778/3476249.3476307. URL https://doi.org/10.14778/3476249.3476307.

[45] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, Oct 1993. ISSN 1573-2878. doi: 10.1007/BF00941892. URL https://doi.org/10.1007/BF00941892.

[46] B. Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, Nov 2016. ISSN 2192-6360. doi: 10.1007/s13748-016-0094-0. URL https://doi.org/10.1007/s13748-016-0094-0.

[47] S. K. Lam, A. Pitrou, and S. Seibert. Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, LLVM '15, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450340052. doi: 10.1145/2833157.2833162. URL https://doi.org/10.1145/2833157.2833162.

[48] N. Laptev, A. Saeed, and B. Youssef. S5 - a labeled anomaly detection dataset, version 1.0(16m), 2015. URL https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70.

[49] A. Lavin and S. Ahmad. Evaluating real-time anomaly detection algorithms – the numenta anomaly benchmark. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), pages 38–44, 2015. doi: 10.1109/ICMLA.2015.141. URL https://doi.org/10.1109/ICMLA.2015.141.

[50] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. *A Comparative Study of Anomaly Detection Schemes in Network Intrusion Detection*, pages 25–36. Proceedings. Society for Industrial and Applied Mathematics, May 2003. ISBN 978-0-89871-545-3. doi: 10.1137/1.9781611972733.3. URL https://doi.org/10.1137/1.9781611972733.3. 0.

[51] D. Lee. Implementation of robuststl. URL https://github.com/LeeDoYup/RobustSTL.

[52] B. Lim and S. Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 379(2194):20200209, 2021. doi: 10.1098/rsta.2020.0209. URL https://royalsocietypublishing.org/doi/abs/10.1098/rsta.2020.0209.

[53] G. Lindgren. *Stationary Stochastic Processes : Theory and Applications*. CRC Press LLC, New York, 1st edition, 2012. ISBN 9781466557802. doi: 10.1201/b12171. URL https://doi.org/10.1201/b12171.

[54] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422, 2008. doi: 10.1109/ICDM.2008.17.

[55] J. Ma and S. Perkins. Time-series novelty detection using one-class support vector machines. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 3, pages 1741–1745 vol.3, 2003. doi: 10.1109/IJCNN.2003.1223670.

[56] G. Mahalakshmi, S. Sridevi, and S. Rajaram. A survey on forecasting of time series data. In *2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16)*, 2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16), pages 1–8, 2016. doi: 10.1109/ICCTIDE.2016.7725358. URL https://doi.org/10.1109/ICCTIDE.2016.7725358.

[57] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, 34(4):802–808, 2018. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2018.06.001. URL https://www.sciencedirect.com/science/article/pii/S0169207018300785.

[58] S. Makridakis, E. Spiliotis, and V. Assimakopoulos. M5 accuracy competition: Results, findings, and conclusions. *International Journal of Forecasting*, 38(4):1346–1364, 2022. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2021.11.013. URL https://www.sciencedirect.com/science/article/pii/S0169207021001874. Special Issue: M5 competition.

[59] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings of the 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, pages 89–94, 2015. URL https://www.esann.org/sites/default/files/proceedings/legacy/es2015-56.pdf.

[60] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai. Kitsune: An ensemble of autoencoders for online network intrusion detection. In *Network and Distributed Systems Security (NDSS) Symposium*, 2018. doi: 10.14722/ndss.2018.23204. URL http://dx.doi.org/10.14722/ndss.2018.23204. Appears in Network and Distributed Systems Security Symposium (NDSS) 2018; Conference date: 18-02-2018 Through 21-02-2018.

[61] Y. Mishura and G. Shevchenko. *Theory and Statistical Applications of Stochastic Processes*. John Wiley & Sons, Ltd, 2017. ISBN 9781119441601. doi: 10.1002/9781119441601. URL https://doi.org/10.1002/9781119441601.

[62] A. Nielsen. *Practical Time Series Analysis*. O'Reilly Media, Inc., Oct 2019. ISBN 9781492041658. URL https://www.oreilly.com/library/view/practical-time-series/9781492041641/.

[63] T. pandas development team. pandas-dev/pandas: Pandas, Feb. 2020. URL https://doi.org/10.5281/zenodo.3509134.

[64] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[65] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[66] S. M. Ross. *Introduction to Probability and Statistics for Engineers and Scientists (Fifth Edition)*. Academic Press, Boston, 5th edition, 2014. ISBN 978-0-12-394811-3. doi: 10.1016/C2013-0-19397-X. URL https://doi.org/10.1016/C2013-0-19397-X.

[67] S. Schmidl, P. Wenig, and T. Papenbrock. Anomaly detection in time series: A comprehensive evaluation. *Proc. VLDB Endow.*, 15(9):1779–1797, jul 2022. ISSN 2150-

8097. doi: 10.14778/3538598.3538602. URL https://doi.org/10.14778/3538598.3538602.

[68] S. Seabold and J. Perktold. statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

[69] M. J. Smith, C. Sala, J. M. Kanter, and K. Veeramachaneni. The machine learning bazaar: Harnessing the ml ecosystem for effective system development. *arXiv e-prints*, art. arXiv:1905.08942, 2019.

[70] M. Sokolova and G. Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009. ISSN 0306-4573. doi: https://doi.org/10.1016/j.ipm.2009.03.002. URL https://www.sciencedirect.com/science/article/pii/S0306457309000259.

[71] V. S. Spelmen and R. Porkodi. A review on handling imbalanced data. In *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT), pages 1–11, 2018. doi: 10.1109/ICCTCT.2018.8551020. URL https://doi.org/10.1109/ICCTCT.2018.8551020.

[72] R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11(4):341–359, Dec 1997. ISSN 1573-2916. doi: 10.1023/A:1008202821328. URL https://doi.org/10.1023/A:1008202821328.

[73] Y. Su, Y. Zhao, C. Niu, R. Liu, W. Sun, and D. Pei. Robust anomaly detection for multivariate time series through stochastic recurrent neural network. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2828–2837, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362016. doi: 10.1145/3292500.3330672. URL https://doi.org/10.1145/3292500.3330672.

[74] L. Tang, H. Lv, F. Yang, and L. Yu. Complexity testing techniques for time series data: A comprehensive literature review. *Chaos, Solitons & Fractals*, 81:117–135, 2015. ISSN 0960-0779. doi: https://doi.org/10.1016/j.chaos.2015.09.002. URL https://www.sciencedirect.com/science/article/pii/S0960077915002817.

[75] N. Tatbul, T. J. Lee, S. Zdonik, M. Alam, and J. Gottschlich. Precision and recall for time series. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.,

2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/8f468c873a32bb0619eaeb2050ba45d1-Paper.pdf.

[76] M. Thill, W. Konen, and T. Bäck. Markusthill/mgab: The mackey-glass anomaly benchmark, apr 2020. URL https://doi.org/10.5281/zenodo.3760086.

[77] J. F. Torres, D. Hadjout, A. Sebaa, F. Martínez-Álvarez, and A. Troncoso. Deep learning for time series forecasting: A survey. *Big Data*, 9(1):3–21, Feb 2021. ISSN 2167-6461. doi: 10.1089/big.2020.0159. URL https://doi.org/10.1089/big.2020.0159.

[78] S. Tuli, G. Casale, and N. R. Jennings. Tranad: Deep transformer networks for anomaly detection in multivariate time series data. *Proc. VLDB Endow.*, 15(6): 1201–1214, feb 2022. ISSN 2150-8097. doi: 10.14778/3514061.3514067. URL https://doi.org/10.14778/3514061.3514067.

[79] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[80] K. Veeramachaneni, C. Sala, H. Dominguez, P. Valentinov, I. Tinawi, R. Diez, K. Wang, V. Coykendall, M. Kanter, S. N. Wong, and B. Farris. MLPrimitives. URL https://github.com/MLBazaar/MLPrimitives.

[81] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17: 261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[82] Q. Wen, J. Gao, X. Song, L. Sun, H. Xu, and S. Zhu. Robuststl: A robust seasonal-trend decomposition algorithm for long time series. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):5409–5416, Jul. 2019. doi: 10.1609/aaai.v33i01.33015409. URL https://doi.org/10.1609/aaai.v33i01.33015409.

[83] Q. Wen, Z. Zhang, Y. Li, and L. Sun. Fast robuststl: Efficient and robust seasonal-

trend decomposition for time series with complex patterns. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 2203–2213, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403271. URL https://doi.org/10.1145/3394486.3403271.

[84] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. doi: 10.25080/Majora-92bf1922-00a.

[85] R. Wu and E. J. Keogh. Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress. *IEEE Transactions on Knowledge and Data Engineering*, 35(3):2421–2429, 2023. ISSN 1558-2191. doi: 10.1109/TKDE.2021.3112126. URL https://doi.org/10.1109/TKDE.2021.3112126.

[86] J. Xu, H. Wu, J. Wang, and M. Long. Anomaly transformer: Time series anomaly detection with association discrepancy. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=LzQQ89U1qm_.

[87] A. Zeng, M. Chen, L. Zhang, and Q. Xu. Are transformers effective for time series forecasting?, 2022. URL https://arxiv.org/abs/2205.13504.

[88] C. Zhang, T. Zhou, Q. Wen, and L. Sun. Tfad: A decomposition time series anomaly detection architecture with time-frequency analysis. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, page 2497–2507, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392365. doi: 10.1145/3511808.3557470. URL https://doi.org/10.1145/3511808.3557470.

[89] Y. Zhao, Z. Nasrullah, and Z. Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019. URL http://jmlr.org/papers/v20/19-011.html.

A | Statistics and probability

This appendix contains a list of formal definitions needed for the definitions introduced in the body of the thesis. The bibliography of this section considers [66] as main source for statistics and probability, and [9, 12, 24, 29, 35] for probability.

A.1. Statistics: definitions

Definition A.1.1 (Population)**:** *We call population a set X representing a collection of elements.*

Definition A.1.2 (Sample)**:** *Given a population X, we call sample any subset $Y \subset X$.*

Definition A.1.3 (Dataset)**:** *We call dataset a function $D : V \to \mathbb{N}$, and we call its domain elements values and the images frequencies.*

Definition A.1.4 (Sample mean)**:** *Let X be a collection of values with $|X| = n$. We call sample mean the following quantity:*

$$\overline{x} = \frac{1}{n} \sum_{x \in X} x \tag{A.1}$$

Definition A.1.5 (Sample median)**:** *Let V be a collection of values, O an arrangement of V with values in increasing order, and $|O| = n$. If n is odd, the sample median is the element at position $(n+1)/2$; if n is even, the sample median is the mean between the elements at position $n/2$ and $n/2+1$.*

Definition A.1.6 (Sample mode)**:** *Let D be a dataset whose values are v_i for $0 \leq i \leq n$ and respective frequencies are f_i for $0 \leq i \leq n$. The sample mode is the set of all elements v_i for $i \in I : f_i = \max_j f_j$.*

Definition A.1.7 (Sample variance)**:** *Let X be a collection of values with $|X| = n$ and $\overline{x}$ the sample mean of X. We call sample variance the quantity:*

$$s^2 = \frac{1}{n-1} \sum_{x \in X} (x - \overline{x})^2 \tag{A.2}$$

Definition A.1.8 (Sample standard deviation)**:** *Let X be a collection of values with $|X| = n$ and $\overline{x}$ the sample mean of X. We call sample variance the quantity:*

$$s = \sqrt{\frac{1}{n-1} \sum_{x \in X} (x - \overline{x})^2} \tag{A.3}$$

Definition A.1.9 (Sample percentile)**:** *Let V be a collection of numerical values and $0 \leq p \leq 1$ the percentage. We call sample $100p$ percentile the data value $v \in V$ such that at least $100p$ percent of the data are less than or equal to it, and at least $100(1-p)$ are greater than or equal to it. If two data values satisfy these conditions, then the sample $100p$ percentile is the arithmetic average of the two values.*

Definition A.1.10 (First quartile)**:** *We call first quartile of some data the 25^{th} sample percentile.*

Definition A.1.11 (Second quartile)**:** *We call second quartile of some data the 50^{th} sample percentile.*

Definition A.1.12 (Third quartile)**:** *We call third quartile of some data the 75^{th} sample percentile.*

Definition A.1.13 (Interquartile range)**:** *Let q_1 be the first quartile and q_3 be the third quartile. We call interquartile range the quantity $IQ = q_3 - q_1$.*

A.2. Probability: definitions

Definition A.2.1 (Sample space)**:** *Let e be an experiment for which we cannot predict with certainty the outcome. If we know which is the set of the possible outcomes of the experiment, we call it sample space and we identify it with* Ω.

Definition A.2.2 (Event)**:** *Let* Ω *be a sample space and* $\omega \in \Omega$ *the outcome of the experiment. Any subset* $E \in \Omega$ *is called event, and if* $\omega \in E$ *we say that* E *has occurred.*

Definition A.2.3 (Algebra of subsets)**:** *Let* Ω *be a sample space and* $\mathfrak{F} \subseteq \mathcal{P}(\Omega)$. *We say that* $\mathfrak{F}$ *is an algebra over* Ω *if it has the following properties:*

1. $\emptyset \in \mathfrak{F}$
2. $E \in \mathfrak{F} \Rightarrow E^C \in \mathfrak{F}$
3. $E, F \in \mathfrak{F} \Rightarrow E \cup F \in \mathfrak{F}$

Definition A.2.4 (σ-algebra)**:** *Let* Ω *be a sample space and* $\mathfrak{F} \subseteq \mathcal{P}(\Omega)$. *We say that* $\mathfrak{F}$ *is a* σ*-algebra over* Ω *if it has the following properties:*

1. $\emptyset \in \mathfrak{F}$
2. $E \in \mathfrak{F} \Rightarrow E^C \in \mathfrak{F}$
3. $E_1, E_2, ... \in \mathfrak{F} \Rightarrow \cup_{i=1}^{\infty} E_i \in \mathfrak{F}$

Definition A.2.5 (Measurable space)**:** *Let* X *be a set and* A *a* σ*-algebra on* X. *We call the tuple* (X, A) *measurable space.*

Definition A.2.6 (Measure)**:** *Let* (X, A) *be a measurable space. A function* $\mu : A \to \overline{\mathbb{R}}$ *is a measure if it has the following properties:*

- $\mu(\emptyset) = 0$
- $\mu(E) \geq 0$ *for all* $E \in A$
- *Let* $\{E_k\}_{k=1}^{\infty} \in A$ *be a disjoint sequence, that is* $E_i \cap E_j = \emptyset$ *if* $i \neq j$. *Then,* $\mu(\cup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} \mu(E_i)$

Definition A.2.7 (Probability measure)**:** *Let* Ω *be a sample space, and* $(S \subseteq \mathcal{P}(\Omega), \mathfrak{F})$ *be a measurable space. A function* $\mu : \mathfrak{F} \to \overline{\mathbb{R}}$ *is a probability measure*

if it has the following properties:

- $\mu(\Omega) = 1$
- $\mu(E) \geq 0$ *for all* $E \in \mathfrak{F}$
- *Let* $\{E_k\}_{k=1}^{\infty} \in \mathfrak{F}$ *be a disjoint sequence, that is* $E_i \cap E_j = \emptyset$ *if* $i \neq j$. *Then,* $\mu(\cup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} \mu(E_i)$

Definition A.2.8 (Probability space)**:** *Let* Ω *be a sample space,* $(S \subseteq \mathcal{P}(\Omega), \mathfrak{F})$ *be a measurable space, and* P *be a probability measure on* $\mathfrak{F}$. *The tuple* $(S \subseteq \mathcal{P}(\Omega), \mathfrak{F}, P)$ *is called probability space.*

Definition A.2.9 (Conditional probability)**:** *Let* $(S, \mathfrak{F}, P)$ *be a probability space, and* $F \in \mathfrak{F}$ *be an event such that* $P(F) > 0$. *For each event* $E \in \mathfrak{F}$, *we call conditional probability of* E *given* F *the following quantity:*

$$P(E|F) := \frac{P(E \cap F)}{P(F)} \tag{A.4}$$

Definition A.2.10 (Independent events)**:** *Let* $(S, \mathfrak{F}, P)$ *be a probability space. Given* $E_1, E_2, ..., E_n \in \mathfrak{F}$, *we say that* $E_1, E_2, ..., E_n$ *are independent, if for each* $\{h_1, h_2, ..., h_k\} \subseteq \{1, 2, ..., n\}$ *with* $k \geq 2$ *we have:*

$$P\left(\bigcap_{i=1}^{k} E_{h_i}\right) = \prod_{i=1}^{k} P(E_{h_i}) \tag{A.5}$$

Definition A.2.11 (Conditional independence)**:** *Let* $(S, \mathfrak{F}, P)$ *be a probability space, and* $A, B, C \in \mathfrak{F}$. *We say that* A *is conditionally independent from* C *given* B *if:*

$$P(A|B, C) = P(A|B) \tag{A.6}$$

Definition A.2.12 (Random variable)**:** *Let* $(S, \mathfrak{F}, P)$ *be a probability space. We call random variable a function* $X : S \to \mathbb{R}$, *such that for each* $x \in \mathbb{R}$:

$$\{X \leq x\} := \{s \in S | X(s) \leq x\} \in \mathfrak{F} \tag{A.7}$$

Definition A.2.13 (Cumulative distribution function)**:** *Let X be a random variable defined over the probability space $(S, \mathfrak{F}, P)$. We call Cumulative Density Function of X the function $F_X : \mathbb{R} \to [0, 1]$, defined as:*

$$F_X(x) := P(X \leq x) \tag{A.8}$$

Definition A.2.14 (Discrete random variable)**:** *Let X be a random variable defined over the probability space $(S \subseteq \mathcal{P}(\Omega), \mathfrak{F}, P)$. We call X discrete random variable if there exist a countable set $C \subseteq \mathcal{P}(\Omega)$ such that $P(X \in C) = 1$.*

Definition A.2.15 (Probability mass function)**:** *Let X be a discrete random variable defined over the probability space $(S, \mathfrak{F}, P)$. We call Probability Mass Function of X the function $p_X(x) := P(X = x)$*

Definition A.2.16 (Bernoulli distribution)**:** *Let X be a discrete random variable and $p \in [0, 1]$ a real number. We say that X has a bernoulli distribution if its Probability Mass Function is:*

$$p_X(x) = \begin{cases} p & \text{if } x = 1 \\ 1 - p & \text{if } x = 0 \end{cases} \tag{A.9}$$

We denote this distribution by $X \sim Be(p)$.

Definition A.2.17 (Continuous random variable)**:** *Let X be a random variable defined over the probability space $(S \subseteq \mathcal{P}(\Omega), \mathfrak{F}, P)$. X is a continuous random variable if there exist a function $f_X : \overline{\mathbb{R}} \to \overline{\mathbb{R}}_+$ such that its Cumulative Density Function can be written as:*

$$F_X(x) = \int_{-\infty}^{x} f_X(x)dx \tag{A.10}$$

We call f_X Probability Density Function.

Definition A.2.18 (Expected value of discrete random variable)**:** *Let X be a discrete random variable taking value in S, and let p_X be its PMF. The expected*

value (or mean) of X is defined as:

$$E[X] = \sum_{x \in S} x p_X(x) \tag{A.11}$$

provided that the series converges absolutely. In this case we say that X has finite expectation. If $\sum_{x \in S} |X| p_X(x)$ diverges, then we say that X has no finite expectation.

Definition A.2.19 (Expected value of continuous random variable)**:** *Let X be a continuous random variable and f_X be its PDF. The expected value (or mean) of X is defined as:*

$$E[X] = \int_{-\infty}^{+\infty} x f_X(x) dx \tag{A.12}$$

provided that the integral converges absolutely. In this case we say that X has finite expectation. If $\int_{-\infty}^{+\infty} |x| f_X(x) dx$ diverges, then we say that X has no finite expectation.

Theorem A.2.1: *Let X be a random variable. Any function $\varphi(x)$ defines a new random variable $\varphi(X)$. If $\varphi(X)$ has finite expectation, then:*

$$E[\varphi(X)] = \begin{cases} \sum_x \varphi(x) p_X(x) & \text{if } X \text{ is discrete} \\ \int_{-\infty}^{+\infty} \varphi(x) f_X(x) dx & \text{if } X \text{ is continuous} \end{cases} \tag{A.13}$$

Definition A.2.20 (R^{th} moment)**:** *Let X be a random variable. If X^r has finite expectation, we call $E[X^r]$ the r^{th} moment of X.*

Definition A.2.21 (Variance)**:** *Let X be a random variable with second moment $E[X^2]$. We call variance the following number:*

$$Var[X] = E[(X - E[X])^2] \tag{A.14}$$

Definition A.2.22 (Standard deviation)**:** *Let X be a random variable. We call standard deviation the number $Std[X] = \sqrt{Var[X]}$.*

Definition A.2.23 (Covariance)**:** *Let X, Y be random variables defined over the same sample space. We call covariance the number:*

$$Cov[X, Y] = E[(X - E[X])(Y - E[Y])] \tag{A.15}$$

B | Code notation

This appendix contains the notation used in chapters to refer to code elements. The development follows an OOP approach strongly based on loosely coupled packages. The following notation is adopted:

- `package name`: the monospaced font is used to refer to a package with name `package name`, e.g., `algorithm` is the way in which the package named algorithm is referred.
- package elements: the blue color is used to refer to objects of a package, e.g., pipelines refer to pipelines defined in `pipelines` (the name of the element may be singular).
- object: the dark green color is used to refer to a specific object of the library, e.g., Pipeline refers to the class named Pipeline (located in `pipelines`).

List of Figures

List of Algorithms

Listings

List of Tables

List of Symbols

Variable	Description
$\mathbb{R}_+$	non-negative real numbers
$\overline{\mathbb{R}}$	real numbers extended with $+\infty$ and $-\infty$
$\phi(x)$	standard gaussian Probability Density Function
$\Phi(x)$	standatd gaussian Cumulative Density Function

Acknowledgements

Versione italiana (english below):

Prima di tutto, voglio ringraziare il Politecnico di Milano per avermi formato come Ingegnere Informatico e tutto il corpo docente. La qualità dell'insegnamento e la severità mi ha permesso di diventare chi sono oggi, e ne sono felice. Poi, vorrei ringraziare il mio relatore di tesi, il Professore Piero Fraternali. Durante la tesi ha saputo suggerirmi e aiutarmi: se andavo nella giusta direzione mi confermava la correttezza, altrimenti mi aiutava a correggere il tiro. Per finire con il personale dell'università, ci tengo a ringraziare i dottorandi del laboratorio, specialmente Nicolò Oreste Pinciroli Vago.

Ora, ci tengo a ringraziare tutte le persone che al di fuori dell'università mi sono state vicine. Prima di tutto, voglio ringraziare mia madre che ha sempre creduto in me, senza mai mostrare un attimo di titubanza nelle mie capacità, e che purtroppo non ha vissuto abbastanza per vedermi riuscire in questo fantastico risultato. Poi voglio ringraziare mio padre nel difficile ruolo che ha avuto e nella vicinanza che mi ha donato in questo percorso. Voglio ringraziare mia sorella e i miei nonni per l'amore e il supporto che mi hanno dato in questi anni. Infine, voglio ringraziare il resto della mia famiglia che mi è rimasto vicino.

Per ultimi ma non da meno, voglio ringraziare tutti i miei amici. Fra questi, alcuni meritano una menzione speciale. Ringrazio Niccolò e Simone, i miei amici di Roma con cui ho passato bellissimi momenti. Ringrazio Luca, un amico con cui ho passato svariate belle serate. Ringrazio Lorenc, un grande amico con cui ho condiviso molti bei momenti. Ringrazio Hajsen, grandissimo ed insostituibile amico con cui non sono mai mancate le risate. Ringrazio Cristian, amico con cui ho condiviso uscite, vacanze e quasi tutti i corsi al Politecnico. Ringrazio Giorgio, amico con molte passioni in comune, e meno corsi in comune. Ringrazio Etion, amico con cui le risate e i momenti seri in università non sono mai mancati. Ringrazio Giulio, amico che ci ha deliziato con le sue bellissime canzoni, ma che non ha ancora fatto l'album. Ringrazio Mirko, amico conosciuto alla fine del percorso, ma con cui ho condiviso molto. Infine, ultimo ma non da meno, ringrazio Riccardo, l'amico di Bologna con cui ho condiviso tantissimi bei momenti.

English version:

Firstly, I would like to express my gratitude to Politecnico di Milano and the teaching staff for the excellent education I received as a Computer Engineer. The quality of the teaching and the high standards maintained have enabled me to become the person I am today, and I am truly grateful for that. I would also like to extend my sincere thanks to my thesis supervisor, Professor Piero Fraternali. Throughout my thesis work, he provided invaluable guidance and support, reassuring me when I was on the right track and helping me find my way back when I wasn't. Finally, I would like to thank all the PhD students in the laboratory, especially Nicolò Oreste Pinciroli Vago.

Moving on, I would like to acknowledge everyone who stood by me outside of the university. First and foremost, I am deeply grateful to my mother, who always believed in my abilities and never once showed any sign of doubt. Sadly, she passed away before she could see me achieve this fantastic result. I would also like to thank my father, who had a tough role but stood by me through it all. To my sister and grandparents, thank you for your love and unwavering support. Finally, I am grateful to the rest of my family for standing by me during this journey.

Last but certainly not least, I would like to express my heartfelt appreciation to all of my friends who have been an incredible support system throughout this journey. Among them, I would like to give special recognition to a few who have played a particularly significant role. First, I would like to thank Niccolò and Simone, my friends from Rome, with whom I have shared countless unforgettable moments. I also want to thank Luca, a dear friend with whom I have enjoyed many evenings. I am immensely grateful to Lorenc, a wonderful friend with whom I have shared many great times. I owe a debt of gratitude to Hajsen, a dear and irreplaceable friend with whom I have never failed to share a laugh. I would also like to thank Cristian, a friend with whom I have shared many evenings, holidays, and nearly all of my university courses. To Giorgio, my friend with whom I share numerous passions and a few university courses, I am grateful. I would also like to thank Etion, my friend with whom I have shared both laughter and serious moments at university. To Giulio, my friend who has entertained us with his songs, but still has yet to release an album, I extend my heartfelt thanks. I am also grateful to Mirko, a friend I met toward the end of this journey, but with whom I have shared many experiences. Finally, last but not least, I would like to thank Riccardo, my friend from Bologna with whom I have shared countless wonderful experiences.