UNIVERSITÀ DEGLI STUDI DI TRIESTE

XXXVI CICLO DEL DOTTORATO DI RICERCA IN SCIENZE DELLA TERRA, FLUIDODINAMICA E MATEMATICA. INTERAZIONI E METODICHE.

ROBUST ANOMALY DETECTION FOR TIME SERIES DATA IN SENSOR-BASED CRITICAL SYSTEMS

Settore scientifico-disciplinare: INF-01

Dottorando:
Stefano Alberto Russo

Coordinatore:
Prof. Stefano Maset

Supervisore:
Prof Luca Bortolussi

ANNO ACCADEMICO 2023/2024

a papà

Abstract

Critical systems are defined as systems where a malfunction can cause significant economic damage, harm to the ecosystem, or endanger human life, potentially resulting in the loss of lives. Financial, Energy, Healthcare, Transportation, Telecommunication, Water Supply, and Defense systems are just some of the more prominent examples of critical systems.

A particular role is played by sensor-based critical systems, which are deeply intertwined between the digital and physical worlds, and where one or more physical quantities are sampled at given time intervals, forming the so called time series, for monitoring and/or control purposes. In both cases, it is important to detect potential anomalies as they may indicate either a malfunction of a sensor (which can lead to apply a control algorithm on unreliable data) or a problem with the system dynamics itself (and thus posing a risk to its integrity).

This thesis work aims at addressing such challenges, and in particular it brings four main contributions: an extensive and critical overview of anomaly detection, a methodological framework for reliably identifying the anomalies in the context of sensor-based critical systems, a software library implementing such methodology, and a real-world case study in the Water Distribution Systems domain.

Contents

Extended abstract

Critical systems are defined as systems where a malfunction can cause significant economic damage, harm to the ecosystem, or endanger human life, potentially resulting in the loss of lives. Financial, Energy, Healthcare, Transportation, Telecommunication, Water Supply, and Defense systems are just some of the more prominent examples of such critical systems.

A particular role is played by sensor-based critical systems, which are deeply intertwined between the digital and physical worlds, and where one or more physical quantities are sampled at given time intervals, forming the so called time series, for monitoring and/or control purposes. In both cases, it is important to detect potential anomalies as they may indicate either a malfunction of a sensor (which can lead to apply a control algorithm on unreliable data) or a problem with the system dynamics itself (and thus posing a risk to its integrity).

Detecting anomalies in a timely and reliable manner in such systems, which typically operate in "harsh" conditions like humidity, moisture, dust, vibration, shock, etc., and under strict operational requirements, is still an open problem. The main challenges are related to the data quality, the resilience required by the anomaly detection processes, the robustness of the algorithms, and the need to provide clear indications to the operators about the potential anomalies and their severity; while keeping the false positives under control and performing the detection in a timely manner, usually in real-time or near real-time.

For example, it is very common for such systems to show several data losses, due to the above mentioned harsh conditions. An anomaly detection pipeline not taking into account such data losses would lead to several false positives, and cause to inevitably loose operator's trust. Similarly, arbitrarily setting thresholds to discriminate normal versus anomalous data instances would not necessarily reflect the true operational mode of the systems, thus leading either to under-estimate or over-estimate the anomalies. Also, using simple binary indications (as anomalous / not anomalous) would not allow to give the operators a sense of the severity of the anomaly, thus not allowing to calibrate (or prioritize) their investigation and/or response. On the other hand, a continuous yet uncapped anomaly score would have the same effect, if not worse. Lastly, anomaly detection must be capable of detecting anomalies as soon as they show themselves. However, many anomaly detection studies and techniques assume to work in a static scenario, where all the data is available for inspection and

there is no incoming data flow.

This thesis work aim at addressing such challenges, and in particular it brings four main contributions: an extensive and critical overview of anomaly detection, a methodological framework for reliably identifying the anomalies in the context of sensor-based critical systems, a software library implementing such methodology, and a real-world case study in the Water Distribution Systems domain.

Anomaly detection is a broad concept, and while the meaning of "anomaly" might look as self-explanatory at first sight, when it has to be treated in a more formal way it immediately reveals its ill-defined nature. This is rooted in the intrinsic subjectivity of the notion of "normality", which is hard to pinpoint and that cause in turn the ill-definition of the term "anomaly". One of the most widely accepted definitions of anomalies is "an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism". This definition was originally provided by Hawkins in 1980, and formulated with respect to outliers (a term that, while intimately entangled with anomalies, identifies yet another concept). However, both terms share being ill-defined: how much is "so much", and what is "suspicion", exactly? They key to robust anomaly detection lies in fixing such two terms, so that ambiguity is removed as much as possible.

Anomaly detection can be approached in three main ways: as a supervised, unsupervised or semi-supervised problem. In the supervised approach, which has ho be intended in its broader form, anomalies are assumed to be already known. Most recent literature excludes such approach from anomaly detection, as it should be framed rather as a classification or pattern recognition problem, and so does this thesis work. If approaching anomaly detection in an unsupervised way instead, no prior knowledge about the anomalies is assumed at all, and the anomaly detection algorithms are challenged to find the most anomalous data instances within a dataset autonomously. While very powerful for data mining applications, in a real time or near real-time setting this approach has intrinsic limitations, mainly related to the impossibility of assessing the anomaly degree of the data instances in absolute terms, which consequentially makes it hard to raise meaningful alarms. Semi-supervised anomaly detection does not make any assumptions about anomalies, but makes a strong assumption about what normality looks like. This allows to underpin the anomaly detection process to a well known notion of normality, thus making it much more robust. The downside of this approach is to require a "normal" dataset, however, in the context of critical systems, it is relatively easy to obtain such dataset because of the commissioning period these systems are usually subject to, where they are closely followed by domain experts to ensure proper and smooth operations. For these reasons, the semi-supervised anomaly detection approach is the approach chosen for this thesis work.

Regardless of the approach, there are many techniques (or algorithms) for anomaly detection. Some treat anomalies as outliers, others as isolated or "discording" data instances in a given metric, and yet other ones as data instances

that, given a model of normality, do not match the predictions. This last class of techniques is also know as model-based anomaly detection, and has two main benefits over the others, in particular with respect to time series data. First, a model allows to capture well the temporal dependency between the observations, which are usually complex to take into account for with other techniques. Second, and in particular with respect to the semi-supervised approach, evaluating the accuracy of the underlying model allows to both calibrate and indirectly assess the robustness of the anomaly detection process itself, which is otherwise hard to evaluate. Directly evaluating an anomaly detection algorithm is indeed challenging because of the intrinsic scarcity of the anomalies, which can lead to potential over-fitting problems towards known anomalies, and that can cause to rely on misleading accuracy metrics. Accuracy metrics computed in this way indeed take into account only known anomalies, giving little insights about how a given anomaly detection algorithm would perform when it comes to truly unseen anomalies, which are likely the most interesting ones.

The methodology for anomaly detection proposed in this thesis work is thus based on semi-supervised, model-based anomaly detection. Moreover, given the requirement of performing anomaly detection in a timely manner, in real-time or near real-time, the natural choice is to employ forecasting models, where no information about future events is used.

A key step of the proposed methodology is to define a so called anomaly index, which quantifies the magnitude of the anomalies in a 0-1 range, for each time step of the time series. Such index, which is defined in terms of the adherence of an observation to a given model of normality, is based (and calibrated) on the model's error probability distribution. When zero, the anomaly index indicates no suspicion of anomaly at all; when one, it indicates an (almost) certain anomaly; while in between, it represents different magnitudes of the potential anomalies. Besides providing a clear indication of the degree of suspicion of an anomaly, the anomaly index also allows to compare the output of different models, being normalized to the 0-1 range. This is particularly relevant not only when working with ensemble models, but also with multivariate time series, provided that using the same model to predict more than one quantity can be seen as an ensemble model itself. In other words, in multivariate time series, the proposed methodology assigns an anomaly index to each of the variables, and for each model used. Merging the various anomaly indexes can then follow various strategies, the two extremes being performing a maximum operation (thus looking for cases where only one of the models of the ensemble spotted the anomaly), or the minimum (thus asking for global agreement between all the models of the ensemble).

To implement the proposed methodology, and to address the challenges relative to the data quality and the resiliency, a relevant part of this thesis work was focused on the development of a software library: Timeseria. This is an object-oriented time series processing library implemented in Python, which in general aims at making it easier to manipulate time series data and to build

statistical and machine learning models on top of it. Unlike common data analysis frameworks, Timeseria builds up from well defined and reusable logical units (objects), which can be easily combined together in order to ensure a high level of consistency. Thanks to this approach, it can address by design several non-trivial issues often underestimated, such as handling data losses, non-uniform sampling rates, differences between aggregated data and punctual observations, time zones, daylight saving times, and more. Timeseria was developed following a set of software development best practices as modularization, unit testing and containerization to ensure portability and reproducibility, and it is available on GitHub and PyPI. In particular with respect to this thesis work, Timeseria implements model-based anomaly detection using the anomaly index introduced above, and it was used to test the proposed methodology on a real world case study in the Water Distributions Systems domain.

Water Distributions Systems (WDS) are a core part of modern city infrastructures, providing a constant water flow from facilities as treatment plants and wells to the final users. Over this path, many components are involved and subject to potential failures, as pipes, valves, pumps, storage tanks, and more. Leakages and breakages are the two most common type of failures, the second being a direct cause of the first one, if small enough not to be disruptive.

A leakage in an underground pipe that goes unnoticed does not only mean wasting a precious resource as water is, but can also cause erosion to the point of creating voids in the above terrain. If such erosion occur in a urban context, roads can collapse and building foundations can shift, posing a severe risk to the population. In case of sudden breakages instead, part of the buildings in the surroundings can be left without water, which is particularly problematic should one of these buildings be a critical infrastructure itself, as an hospital.

Over the course of recent years, more and more WDS have been equipped with remote monitoring systems for a number of reasons, including to better understand their dynamic, perform optimisations, and schedule maintenance tasks. However, many issues can arise with the sensors, in particular given the harsh environment in which they operate. Sensors not working properly can jeopardize such initiatives, and lead to unusable historical data.

It is therefore clear that having a robust anomaly detection process in place that operates in a timely manner can bring significant benefits for such systems, both in terms of detecting sensor issues and failures in the system itself, thus allowing to promptly address them.

The case study considered in this thesis work consisted of 14 nodes of a WDS in the Friuli Venezia Giulia region, in northern Italy. The dataset comprised flow rate and pressure sensor measurements for each node, sampled at six-minute intervals. The dataset showed several data losses, all detected and marked as such with the Timeseria library, which was used for data preprocessing and to perform model-based anomaly detection, according to the proposed methodology. Several forecasting models have been considered as candidates for model-based anomaly detection. The choice fell on a set of deep learning models, because of their ability to capture complex nonlinear depen-

dencies. Within the deep learning models suitable for time series forecasting, the best compromise was then to use a Long Short Term Memory neural network. To capture both the intrinsic correlation between the pressure and the flow rate and their typical patterns in a separate way, a key step for the anomaly detection process, an ensemble of two types of models was used. The first was trained to predict the value for the next time step given only a window of past data (the previous 24 hours), in order to capture the patterns. The second was instead trained in order to capture the correlations. More in detail, such model was trained to predict the value of a target quantity (e.g. flow rate) at the next step given the contextual data at the same next time step (e.g. pressure). It also used a much smaller window of past data (1 hour) to give some context to the model but not enough to capture any pattern. Issues with the dynamics of the WDS are most likely to be spotted by the first model, while issues with the sensors by the second one. The outcome of the anomaly detection performed with these two type of models where then merged together.

This case study showed how the methodology proposed in this thesis work can be applied in a real world scenario in order to perform robust anomaly detection from start to end. Trivial anomalies (as planned pipe closures) could all be successfully identified with great confidence, while some more subtle and nontrivial anomalies, often symptoms of upcoming issues as sensor failures, could be identified as well. The anomaly index defined according to the proposed methodology allowed differentiating such anomalies based on their magnitude, thus enabling to prioritize their investigation. Lastly, no significant false positives were found, as per intrinsic design of the proposed methodology, which was one of they key requirements for implementing robust anomaly detection in the context of critical systems.

Acknowledgments

This work has been supported by the Italian Research Center on High Performance Computing, Big Data and Quantum Computing (ICSC). , an initiative funded by the European Union - NextGenerationEU - and the Italian National Recovery and Resilience Plan (NRRP) - Mission 4, Component 2, without which this work would not have been possible.

The case study presented in chapter 6 of this thesis was made possible through the invaluable support of Idrostudi s.r.l., an Italian engineering firm specializing in civil and environmental engineering with a focus on water resources management. Their provision of a real-world, comprehensive dataset enabled a realistic and informed case study, while the domain expertise shared by Idrostudi's team enriched the research with domain-specific insights, enhancing the practical relevance and applicability of this work.

List of figures and tables

Figures

Tables

Chapter 1

Introduction

Critical systems are defined as systems where a malfunction can cause significant economic damage, harm to the ecosystem, or endanger human life, potentially resulting in the loss of lives. Financial, Energy, Healthcare, Transportation, Telecommunication, Water Supply, and Defense systems are just some of the more prominent examples of critical systems.

In general, critical systems operate under high-stakes, high-risk conditions, where even minor failures can lead to significant consequences, and have therefore to be protected in the best possible way.

Figure 1.1: An artistic representation of various critical systems. Image by OpenAI DALL-E.

In 2010, the Deepwater Horizon oil rig suffered from a major failure. It caused the death of 11 workers, and polluted the gulf of Mexico with almost a billion liters of oil , with unquantifiable long term damage to the ecosystem. Economic consequences on fishing and tourism were also severe, besides the cost of the platform itself of about 560 million US dollars. British Petroleum (BP) had to pay over $60 billion US dollars in fines, settlements, and cleanup costs. As of today, the Deepwater Horizon platform is one of the major disasters in recent times, and the largest marine oil spill in history.

Figure 1.2: The Deepwater Horizon platform in flames after the blowout.

After the disaster, a common question emerged among experts: *"how could it possibly have happened?"*. Countless investigations have been carried out, given the impact of the failure, and the causes narrowed down to a single root factor that started the chain of catastrophic events: a pressure test whose results were misinterpreted by the operators (Figure 1.3).

The official aftermath report [55] states that *"the real time drill pipe pressure started dropping, which indicates a leak bleeding off the pressure somewhere"* and that the investigators found *"no evidence that the crew was aware of this indicator"*. In other words, the test showed unusual pressure discrepancies, that the crew misinterpreted and considered as still normal.

We will never know if the Deepwater Horizon disaster could have been avoided with modern anomaly detection techniques. What we do know, however, is that any effort towards building systems that could prevent it from happening again is a step forward.

In the context of critical systems, a particular role is played by the so called sensor-based critical systems, where one or more physical quantities are sampled at given time intervals for monitoring and/or control purposes. In

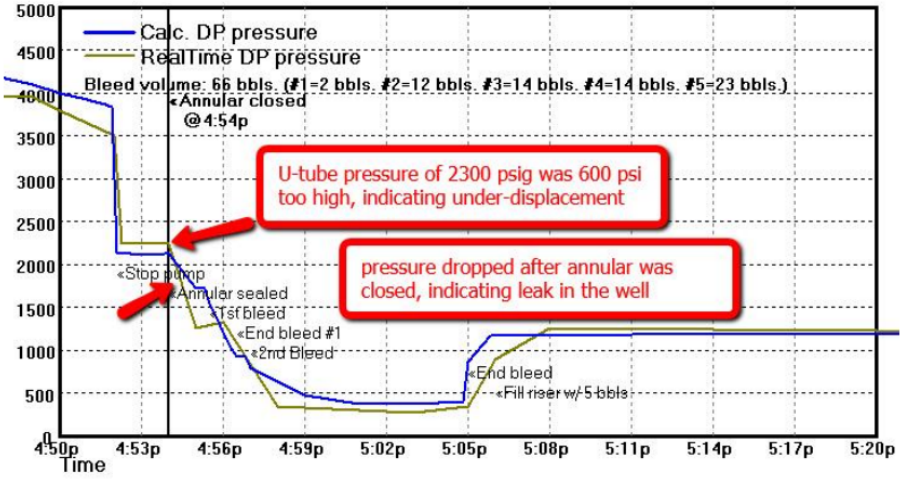


Figure 1.3: Drill pipe pressures during negative pressure test, few hours before the blowout.

both cases, it is important to detect potential anomalies as they may indicate either a malfunction of a sensor (which can lead to apply a control algorithm on unreliable data) or a problem with the system dynamics itself (and thus posing a risk to its integrity, as in the case of the Deepwater Horizon).

Having a robust anomaly detection process in place, that operates in a timely manner, can bring significant benefits to such systems, both in terms of detecting sensor issues and system failures, thus allowing to promptly address them.

Anomaly detection is a broad concept, and while the meaning of "anomaly" might look as self-explanatory at first sight, when it has to be treated in a more formal way it immediately reveals its ill-defined nature. This is rooted in the intrinsic subjectivity of the notion of "normality", which is hard to pinpoint and that cause in turn the ill-definition of the term "anomaly".

One of the most widely accepted definitions of anomalies is "an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism". This definition was originally provided by Hawkins in 1980, and formulated with respect to outliers (a term that, while intimately entangled with anomalies, identifies yet another concept). However, both terms share being ill-defined: how much is "so much", and what is "suspicion", exactly?

While anomaly detection has been tackled extensively over the recent years, the entire field is facing major challenges, mainly because of such underlying ambiguity. A recent article, titled "Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress" [88] provides a quite self-explanatory statement about the current situation.

Over-rewarding metrics, questionable ground truths, and the constant presence of trivial anomalies in the benchmarking datasets all cause a false sense of reliability that ends up in causing several false positives.

In particular with respect to critical systems, any false positive, triggered by either a data loss, an error in a sensor reading, or because of denying such ambiguity instead of attempting to ground it, would inevitably cause to loose operator's trust, with the effect of undermining any effort.

Alerting the operators on a solid basis about the potential anomalies is therefore the first and foremost requirement, and the solution starts by defining what "so much" and "suspicion" mean, so that ambiguity is removed as much as possible.

This thesis work brings four main contributions towards this goal, which can be summarized as:

1. an extensive and critical overview of anomaly detection;
2. a methodological framework for reliably identifying anomalies;
3. a software library implementing such methodology; and
4. a real-world case study in the Water Distribution Systems domain.

Chapter 2

Critical systems

2.1 Taxonomy

Critical systems play a crucial role in modern life and refer to any system, whether hardware, software, or a combination of both, that is essential for the operations of a larger system or organization, and whose failure or malfunction can have serious consequences. Such consequences can include significant economic damage, harm to the ecosystem, or endangering human life, potentially resulting in the loss of lives.

Financial, Energy, Healthcare, Transportation, Telecommunication, Water Supply, and Defense systems are just some of the more prominent examples of such critical systems, which can be sub-divided in four main types [35]:

1. Safety-Critical systems, where failure can lead to loss of life, serious personal injury, or damage to the natural environment;
2. Mission-Critical systems, where failure can lead to the inability of completing the overall objectives;
3. Business-Critical Systems, which are essential for the operation of a business or organization;
4. Security-Critical systems, which if compromised can cause theft or loss of sensitive data, or disruption in enforcing authentication and authorization.

A particular role is played by sensor-based critical systems, which are transversal to such categories, although usually more concentrated in the first two. In sensor-based critical systems, one or more sensors are read at given time intervals, either for monitoring purposes or to trigger specific actions, possibly in a feedback loop mode.

For example, in industrial process control many applications use feedback loops to ensure optimal production conditions: sensors as pressure, temperature, and flow rate are constantly monitored and the system is then adjusted to maintain both efficiency and safety.

The key difference between sensor-based and generic critical systems lies in their physical component, which makes them deeply intertwined between the digital and physical worlds. This kind of entanglements has usually two main implications:

1. the presence of an underlying phenomena obeying physical laws that can be captured, and
2. the intrinsic challenges of measuring physical quantities in a real-world, and possibly harsh, setting.

With respect to a financial critical system as the stock market, the difference should be pretty clear: there is no physical phenomena underlying stock exchanges, and the stock market data is very accurate.

Instead, an air pollution monitoring system can capture specific air dispersion mechanisms, involving several environmental factors as wind, atmospheric pressure and air temperature. Moreover, measuring air pollutants is subject to several technical issues: rain and dust, as well as highly variable temperatures can severely affect the quality of the readings, if not causing the sensors to fail entirely.

By "harsh" conditions, similar settings are usually implied, where issues due to humidity, moisture, dust, vibration, shock, magnetic interference, etc. can severely affect sensor measures. Not all sensor-based critical systems operate in harsh conditions, but most of them do.

2.2 Control and monitoring

Within sensor-based critical systems, "control and monitoring" refers to the continuous process of managing system operations and tracking performance metrics to ensure stability, safety, and adherence to specified operational parameters.

A key aspect when automating this process is to make use of technologies capable of handling the above mentioned harsh conditions, together with strict operational needs, which require specific design choices. Common consumer electronics fall short, leaving space to the so called "rugged" or "industrial" components", which can deliver the required reliability, precision, and fail-safe operations.

However, over the recent years some of these requirements have been progressively relaxed, in particular with respect to monitoring-only applications, in order to benefit from the advances in the Information Technology (IT) space.

2.2.1 SCADA

Supervisory Control And Data Acquisition (SCADA) is a control system architecture placed on top of hardware solutions which are capable of interfacing with sensors and actuators, as the Programmable Logic Controllers (PLCs) [23]. These are industrial computers that has been ruggedized and adapted

for the control of industrial processes, thus providing the standards required for such applications, as resistance to shock and vibration. SCADA systems are used in most industrial processes, as steel making, power generation and distribution, chemistry, etc. and are thus central in critical systems as well.

Historically, SCADA systems are closed systems, limiting the interoperability options to the bare minimum. When connectivity is required, SCADA systems usually make use of dedicated cabled connections, resorting to wireless connectivity only when strictly necessary, using domain-specific technologies.

With the advent of the computer Local Area Networks (LAN), the operating scenario started to change, given the great benefits that these systems could gain from centralizing some common functionalities over the network, as advanced monitoring solutions including anomaly detection.

On the other hand, adopting LAN networks could open the doors towards insecure and public communication channels (i.e. the Internet) and required adopting some security counter-measures, as the so called "air gap". In an air-gapped system, there is a physical separation between the internal (secure) network and any other networks, as the Internet.

True isolation, however, is difficult nowadays due to the massive spread of connectivity [65]. Moreover, taking advantage of most recent innovations in the IT domain (as geographically distributed Virtual Private Networks, centralized cloud infrastructures, Virtual Reality-based remote support, etc.) inevitably requires to break the air gap.

Both vendors and industries have thus been forced to cope with this new reality, and given also the constantly increasing security levels of modern public networks, are progressively relaxing their air gapping policies, thus opening the doors for data interoperability and new opportunities.

2.2.2 Data loggers

Data loggers are systems designed for the solely propose of logging data over time. Usually small and easy to carry, before the digitization era, they were mechanical devices keeping track of data by physical means. For example, a disk rotating under an arm with a print head, which is sensible to the ambient humidity, thus keeping track of it as the disc rotates over time.

As electronics progressed over the last decades of the 20th century, such devices started to operate in a digital way. In this case, measurements from one or more sensors are acquired using an ADC (Analog to Digital Converter) driven by a computing element, and their data stored in an internal memory. Digital data loggers can operate either connected to a power socket or using an internal battery, and can have several channels performing the data acquisition at different bit depths and sampling intervals.

Data acquired by digital data loggers was originally envisioned to be collected manually: an operator had to either bring a computer close to the data logger (or vice-versa) and connect it, or its internal memory had to be plugged into a reading device (as a memory card reader).

Given that this operation was time-consuming and that any failure in the acquisition process would go unnoticed until the dump of the data, in recent times data loggers started to be equipped with various forms of connectivity, thus enabling remote monitoring in real-time or near real-time.

2.2.3 Internet of Things

The term "Internet of Things" (IoT) was first used in 1999 by British technology pioneer Kevin Ashton to describe a system where objects of the physical world could be connected to the Internet using sensors [68]. In its original statement [6], Ashton focused in particular on RFID (Radio Frequency IDentification) tags, which allow to "sense" objects belonging to the physical world by digital means.

Today, the term IoT encompass a wide range of technologies capable of connecting various objects together, not necessarily using sensors. As with every new trend, the term started to be adopted even where other, potentially more well established terms were already in place, as for example Internet-connected the SCADA systems and data loggers.

The modern Internet of Things can be thus everything and nothing: a vending machine proposing discounts on some items because of a surplus in the central warehouse, a fitness tracker connected to a mobile phone, the mobile phone itself, but also a car, or a nuclear power plant.

Arising mainly from the consumer electronics domain, remote monitoring IoT solutions started to overlap with the natural progress of SCADA systems and data loggers, generating considerable confusion. Coining new terms along the way, as IIoT (Industrial Internet of Things), did not help in the process.

In this context, it has to be said that, abuses aside, the IoT paradigm sees the Internet as its core component, while similar technologies not directly using the Internet were available even before. An example is the M2M (Machine-to-machine) communication, which was *"based on closed purpose–built networks and proprietary or industry–specific standards, rather than on Internet Protocol (IP)–based networks and Internet standards"* [68].

The process of providing with connectivity SCADA systems and data loggers started thus long time before the IoT was born, by adopting this kind of solutions. Moreover, moving towards a more standardized approach using a public network as the Internet involves many challenges, as the need of more advanced software functionalities and more powerful computing resources, and most importantly, to properly implement security standards.

Adopting the Internet of Things without without properly enforcing the security aspects can cause severe backfire should the system be compromised, and in the context of critical systems its adoption has been rightfully slowed down and delayed, to allow for the proper addressing of these concerns [53].

As of today, relevant progress has been made, and some critical systems are already equipped with solutions that are technically IoT-based. However, they are limited to use-cases where a system compromise would result in moderate or contained damage.

2.3 Time series data

In the vast majority of cases, sensor-based critical systems operates on time series data, which represents the evolution of the system over time. Not to be confused with generic series data (as text strings) or signals (as an audio waveform), time series have in their time component their main characteristic. In other words, besides the sequential and time-dependent ordering, the specific timestamp with respect to a given reference time system of each observation carries important information as well.

Timestamps allow to compare (e.g. correlate) time series from different sources, either acquired at different locations across the same system or from exogenous origins, and to account for phenomena not directly dependent on the system dynamic. Only a small portion of critical systems indeed operates in isolated conditions, while for most of them environmental and human factors are key parameters when modeling their behavior.

For example, an energy grid will be susceptible to daily patterns, seasonalities and bank holidays; a wind farm to meteorological conditions, and a solar farm to a combination of the time of the day and cloud coverage. Some of these parameters can be inferred from the timestamps of the time series themselves, as the time of the day and the bank holidays, while others, as the meteorological conditions, require to compare different time series.

In both cases, it is important for the time component to be correctly captured and logged. Properly managing the time domain, including taking into account peculiarities as time zones, daylight saving times and calendar arithmetic is a key step for reliably processing sensor-based critical systems data.

An important step in this process is clock synchronization. In general, any clock must be initialized with the correct time, and a good clock must also not drift. The same holds true for SCADA systems, data loggers, and IoT devices. However, many issues can arise in this context: some because of an approximate initial configuration, others because of poor design choices that use relative time instead of absolute time (and thus not allowing to keep track of the timestamps in an effective way), while yet others because of the internal quartz sensibility to temperature variations, power supply instability, or just the aging of components.

In Internet-based systems, an effective way to overcome such issues and achieve both accurate and persistent time synchronization is it to use the Network Time Protocol (NTP), which is is used by hundreds of millions of computers and devices to synchronize their clocks over the Internet. Already widely used in the IoT space, such approach is being progressively adopted also in the context of SCADA systems and data loggers. In case of an air gap, other methodologies can be used, as for example local time synchronisation services or GPS-based time synchronization.

Time series can be in general univariate or multivariate. In univariate time series (Figure 2.1), a single quantity is tracked along the time domain, while in multivariate time series (Figure 2.2) quantities can range from two to an arbitrary number.

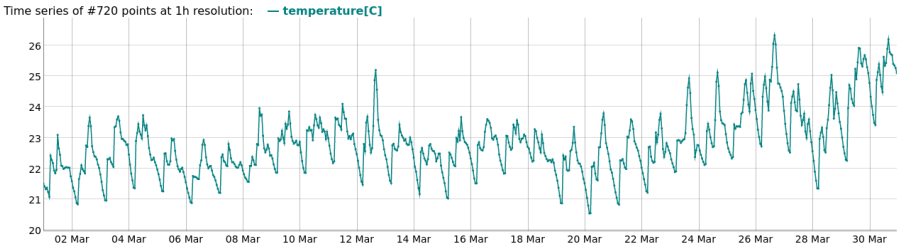


Figure 2.1: Example of a simple univariate time series.

Multivariate time series usually include data from a single acquisition device equipped with more than one sensor, but when there is a need of comparing data from different sources, they can be merged in a single, multivariate time series as well (Figure 2.3) .

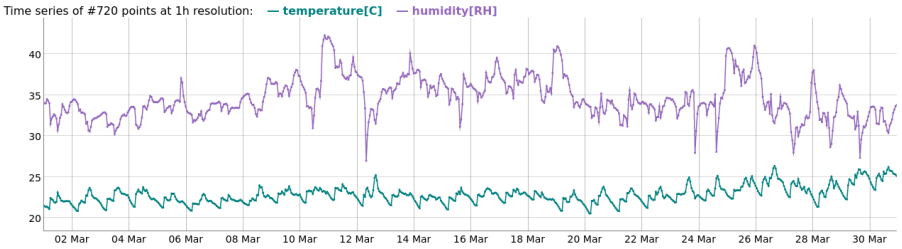


Figure 2.2: Example of a multivariate time series by a single acquisition device.

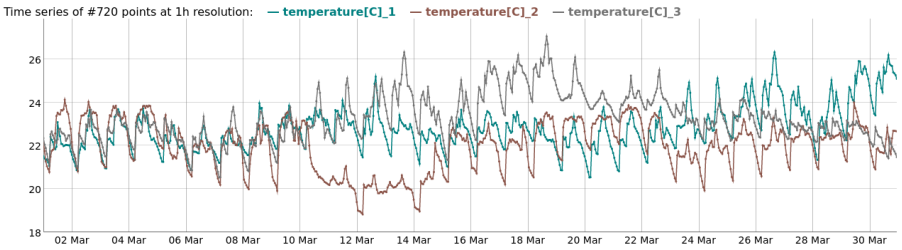


Figure 2.3: Example of a multivariate time series by multiple acquisition devices

Time series can be processed in many ways, and a particularly interesting approach is the so called "streaming mode", where new observations are processed as soon as they come in. In this case, an important consideration is weather history should be allowed to be rewritten, or not.

Such aspect is relevant also in batch processing mode, where time series are processed either as a whole or in smaller batches, but it is more central in streaming mode since this is usually aimed at providing real-time or near real-time insights. In case of out-of-order data transmission, a data point which is received after a most recent one, and that is still processed, can cause to contradict the insights previously delivered. For example, in energy consump-

tion tracking, the maximum peak power can be just overwritten by late-coming observations.

Whether to accept such observations or not is a design choice, which has pros and cons. On the pros side, none of the available data is lost. On the cons side, streaming data processing (where out-of-order data just cannot fit) could say one thing, while a batch (re)processing of the same data could say another.

Similarly to clock synchronization, also for streaming data processing a proper design of the data acquisition process is fundamental for obtaining reliable time series data.

2.4 Anomaly detection

2.4.1 Role

The role of anomaly detection in critical systems is mainly related to avoid costly failures. Broadly speaking, in sensor-based systems there are two main classes of anomalies: *sensor anomalies* and *system anomalies*.

Sensor anomalies are related to malfunctions in the sensing hardware. Such anomalies are the most frequent ones and are particularly relevant whenever one or more actions are going to be undertaken based on the sensor readings (e.g. in feedback loop systems), which can in turn lead to potential issues and damages. Even when there are no actions to be undertaken, not detecting a sensor failure in a timely fashion leads to acquiring garbage data for a potentially long period of time, which can be an issue if collecting historical data, especially for ex-post data analysis and regulatory compliance.

System anomalies are instead related to anomalies in the dynamics of the system itself. These can be due to sub-optimal working conditions, early signs of some component not working properly anymore, or because of a failure. The order of importance is of course progressive, while the ease of detection is inversely proportional: failures are relatively easy to spot, early warnings are harder, and sub-optimal working conditions are the hardest, given their closeness to the normal operational mode.

2.4.2 Challenges

The main challenge in implementing anomaly detection in the context of critical systems lie in the reliability it requires. Since anomalies can trigger high-priority operator responses or investigations, to the point of stopping operations or waking someone up in the middle of the night, false positives are very costly. On one hand because of the economic cost, and on the other, perhaps most importantly, because they can cause operators to loose trust in the anomaly detection process itself. Alerts for potential anomalies must be therefore fired with an high level of confidence.

Unfortunately, the first issue to achieve this goal already arise when acquiring, storing and processing the data, and undermines the main pillar on which an anomaly detection process is to be built: the reliability of the data.

Many issues can affect this step, either because of poor design and architectural choices (which are surprisingly common in the field) or intrinsic hardware limitations and failures due to harsh operating conditions.

For example, storing the timestamps in local time instead of Coordinated Universal Time (UTC), storing placeholder values (as 0 or -99) when a sensor cannot be read, or sending data in an unordered fashion can very easily generate false positives in anomaly detection. Data is also often sampled at unsynchronized or irregular time steps because of missing hardware clocks synchronization, intrinsic limitations (e.g. a single acquisition device that has to sample many sensors in sequential mode), or because of retrying to read a sensor that could not be correctly read after a short amount of time.

Because of this, a resampling step is almost always needed to make the time steps uniform, which is required for nearly all modeling techniques for time series data. During this step, data losses which naturally occur in such systems must be taken into account, that can be particularly challenging if using placeholder values in case of data acquisition errors. Even if assuming that the missing data points can be correctly detected (either because they are completely missing or because they are correctly marked as null values), the resampling step will anyway assign a value to the data points created over such gaps, usually by linear interpolation. If there is no metadata for the data points, after the interpolation step it is impossible to distinguish between the data points rightfully created by the resampling process and the data points which were completely reconstructed.

If not properly handled, the resampling step can therefore easily lead to the so called garbage-in garbage-out paradigm. Fitting or applying a model on (potentially large) portions of fully interpolated data can indeed easily confuse the model, and lead to completely unrealistic predictions, thus causing anomaly alerts to trigger on anomalies that were just nonexistent.

Besides the challenges related to the data processing, there are also several challenges within anomaly detection itself, which is a very noisy research field and hard to navigate. One of the main sources of such noise is to approach anomaly detection with how anomalies will look like already in mind, e.g. as a supervised problem. However, truly relevant anomalies are yet unknown, and targeting known anomalies is not much of use in detecting them. Supervised anomaly detection would also require a balanced dataset between the normal and anomalous data instances, which is clearly unfeasible unless adopting data augmentation or synthetic data generation methodologies, which has several implications to consider.

If instead not using the known anomalies to design, fit or train the the algorithms involved in the anomaly detection process, (e.g. using a semi-supervised or an unsupervised approach), it is common to use the few known anomalies as an evaluation test bed. While widely accepted, this practice has an important drawback almost never discussed, which is to heavily bias the selection towards techniques that will perform well on known anomalies, but not necessarily on unknown ones.

One of the (bold) arguments made in thesis thesis work is that most often than not there are just not enough know anomalies to allow for a data-driven evaluation, and that the focus should be instead directed towards designing anomaly detection techniques that could be, at least partially, assessed or evaluated without knowing anything about the potential anomalies.

This is also particularly relevant for newly built critical systems, where there is just no history of known anomalies, but where spotting yet completely unknown ones has the same (if not more) value than on systems where some historical anomalies are available.

Another challenge in implementing anomaly detection for critical systems is to make it suitable for real-world scenarios. Many studies and techniques are indeed limited to artificial ones, for example where the data is all already collected and/or manually cleaned, there are no time constrains on detecting the anomalies, or just make use of toy datasets, which in the time series domain usually translates into detecting trivial anomalies on univariate time series.

For example, a recent article [49] published by the European Space Agency in 2024, with respect to the potential of Machine Learning techniques for anomaly detection in real-world satellite telemetry, states that the field *"is currently hampered by a lack of comprehensible benchmarks for multivariate time series anomaly detection"*.

Such article also enumerates the requirements for the ideal anomaly detection algorithm, which in the authors opinion does not exist yet, differentiating the "shalls" (a mandatory feature) vs. the "shoulds" (a desirable feature). The first "shall" states that such algorithm *"shall provide a binary response (i.e. 0 – nominal, 1 – anomaly)"*, and it also adds that: *"It is not enough to provide continuous anomaly scores to SOEs*[1]*, so a thresholding mechanism should be a part of the algorithm. A clear boundary is needed to decide if something should be alarmed to operators or not."*.

This requirement is however somewhat unrealistic if taken as-is. As it will be extensively discussed in Chapter 3, anomalies are indeed an ill-defined concept, and it is thus just impossible to provide a binary response whether a data instance is anomalous or not. Moreover, a thresholding mechanism can cause to miss anomalies slightly below the threshold, and does not allow to prioritize operators' response nor investigation, either in terms of urgency or in case of two or more anomalies arising simultaneously. While this might not be a major concern in the satellite operations domain, where there is a 24/7 control room, this is not the case for most of critical systems, where operators have many tasks and need to prioritize their activities and responses.

It is instead of paramount importance to provide a clear boundary to decide if something should be alarmed to operators or not, as the authors say, rather than providing just a meaningless (e.g. arbitrary or uncapped) anomaly score, and this thesis work aim at addressing exactly this need. However, once the alarm is triggered, to restrict the indication about the potential anomaly to just a binary state is rather questionable.

[1] Spacecraft Operations Engineers.

The authors also put strong emphasis on the importance of not triggering false positives, and clearly state how the false positives rate should be a key metric for the successful application of anomaly detection in their context. They state: *"The highest priority aspect relates to the proper identification of anomalous events, but with a strong emphasis on avoiding false alarms at the same time"*. They also add: *"Again, it is of paramount important to avoid false positives. It is strongly preferable to miss some channels rather than to wrongly identify many irrelevant channels."*

Such reasoning on the false positives can be extended to any critical system, and together with the need of providing a clear boundary to decide if something should be alarmed to operators or not, it represents one of the main requirements underpinning this thesis work.

Again with respect to the above mentioned article, the second "shall" states that algorithms *"shall allow for real-time, online, streaming detection"*. The term "real-time" implies to detect anomalies as soon as new data comes in, without any lag. While this might be the case for spacecraft telemetry, such requirement is too strict for the vast majority of critical systems, where a near real-time approach might be sufficient, provided that the detection is performed in a *timely* manner. What "timely" means, it then depends on the use-case.

The two main requirements for anomaly detection in critical systems, generalized to a variety of use-cases and including the considerations presented in this section, can be therefore enumerated as follows:

1. to provide a clear, robust and reliable indication about the potential anomalies and their degree of suspicion, and

2. to work in an online and streaming fashion, performing the detection in a timely manner, usually in real-time or near real-time.

Addressing these two requirements generates a chain of sub-requirements and sub-challenges, including how to define a meaningful anomaly scoring from which to derive such clear indications, how to implement a robust methodology to compute the scoring, how to properly handle data losses in particular with respect to the streaming mode, and many more, which are all core aspects of this thesis work.

Chapter 3

Anomaly detection overview

The goal of this overview is to frame anomaly detection consistently, while acknowledging its ill-defined nature and thus trying to ground the intrinsic ambiguity it comes with.

Definitions, notions and techniques are supported by many references, and in particular by four main reviews, selected as the best structured ones among the large body of work on the topic: two on generic anomaly detection (Chandola [15] and Pang [63]) and two specific on time series anomaly detection (Cook [20] and Blaquez [10]).

3.1 Defining anomalies

3.1.1 A naturally ill-defined concept

Anomalies have been tried to be defined in many ways in scientific literature over the past decades. Often used interchangeably with terms as "outliers", "rare events", "novelties", and more, such elusive concept has been defined in ways such as:

- *An outlying observation, or "outlier", is one that appears to deviate markedly from other members of the sample in which it occurs.* (Grubbs, 1969) [32]
- *An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism* (Hawkins, 1980) [33]
- *Anomalies or outliers are substantial variations from the norm* (Mehrotra, 2020) [48]
- *Anomalies, a.k.a. outliers, are data instances that significantly deviate from the majority of data instances* (Pang, 2021) [63]

- *Anomalies are patterns in the data that do not conform to expected behavior* (Chandola, 2009) [15]
- *Anomalies are patterns in data that do not conform to a well defined notion of normal behavior* (Chandola, 2009) [15]

Before the seventies, most papers discussing outliers and related techniques did not even bother to provide a definition. It was just considered a self-explanatory term. At some point, authors started to feel the need of defining what an outlier is, and as of today, the two most widely used definitions in literature are the first two of the list above, by Grubbs and Hawkins, with a shift towards the definition provided by Hawkins in recent research.

Grubbs refers to a generic "marked deviation", while Hawkins adds more: that such marked deviation cause to arise "suspicions that it was generated by a different mechanism", or in other words, that such observation is from a different population. However, when an outlier is detected and the suspicion arises, whether such deviating observation is generated by a different mechanism or not is yet to be assessed, and all another story. Immediately following his definition, Hawkins indeed writes:

"There are two basic mechanisms which give rise to samples which appear to have outliers. It is a matter of some importance which of the mechanisms generated any particular set of observations since this consideration certainly affects, or should affect, one's subsequent analysis of the data. Mechanism (i): The data come from some heavy tailed distribution such as Student's t. There is no question that any observation is in any way erroneous. Mechanism (ii): The data arise from two distributions. One of these, the 'basic distribution', generates 'good' observations, while another, the 'contaminating distribution', generates 'contaminants'."

In the context of anomaly detection, anomalies are clearly the "contaminants", while "good'" observations are the main source of false positives. Discerning which between the two mechanisms is at the origin of the outliers was central to the the discussion for a good portion of the 20th century, starting with Irwin in 1925 with his paper titled "On a Criterion for the Rejection of Outlying Observations", and revisited by Grubbs in 1969 with his tutorial paper "Procedures for Detecting Outlying Observations in Samples", where he wrote:

"An outlying observation may be merely an extreme manifestation of the random variability inherent in the data. If this is true, the values should be retained and processed in the same manner as the other observations in the sample. On the other hand, an outlying observation may be the result of gross deviation from prescribed experimental procedure or an error in calculating or recording the numerical value. In such cases, it may be desirable to institute an investigation to ascertain the reason for the aberrant value.

The core of outlier analysis (and not just detection) is thus about deciding whether they should be considered as compatible with a statistical fluctuation,

or caused by a different mechanism. However, this aspect seems to have been almost completely marginalized in modern research, where the main investigation area is about the detection, while little effort is put on improving the criteria for determining which of the two above options is more likely.

Outliers are often just implicitly considered as something "to avoid", which while it might work when performing statistical analysis where just removing them is a viable (yet questionable) choice, it is instead counter-productive when the target of the analysis are the outliers themselves.

Part of the problem is that the above mentioned criteria cannot be based on pure mathematical assumptions, but they depend on many factors. Back in 1925, Irwin indeed wrote, with respect to how to set a threshold for deciding whether to include or not an outlier in the analysis: *"It is a matter of opinion which the mathematician cannot settle, how small $P(\lambda)$ must be before it is justifiable to reject the observation"*.

As of today, such question remains largely unanswered. In a way, we got very good in detecting outliers, but we still fall short when it comes to deciding what we should do with them: if to treat them as normal, or if they represent an *anomaly*.

With respect to anomalies, when not used synonymously with outliers (and section 3.1.2 will argument why this should not be the case), these are defined in a broader sense, usually involving the concept of an "expected behavior". Interestingly enough, the last two definitions in the list at the top of this section are used by the same authors within the same review. However, they convey two very different concepts: the first refers to a generic "expected behavior", while the second to a "well-defined notion of normal behavior".

While this seems a subtle difference, it actually has big implications. If we talk about an "expected behavior", then the question should arise quite spontaneously: the expected behavior for *who*? If we instead have a "well defined notion of normal behavior", then we have much less ambiguity in the definition of what is normal and what is not.

Again with respect to the last two definitions, they also bring into play yet another concept: the *pattern*. Patterns implicitly require a structure or relationship within the data to exist, while anomalies can be just single observations in a dataset where there are no structures or relationships at all: a person 2.8 meters tall would clearly be an anomaly, no patterns involved.

These are just some examples of the confusion surrounding the terminology in this field, and just the tip of the iceberg. Most research works and published papers keep mixing and matching different concepts, definitions and terminology, often taking for granted that other people working in the same domain will implicitly assume the same meaning.

While this practice does not necessarily mean that such works are low quality, it does make the landscape quite hard to navigate, especially when it comes to understand benefits and limitations of the various techniques.

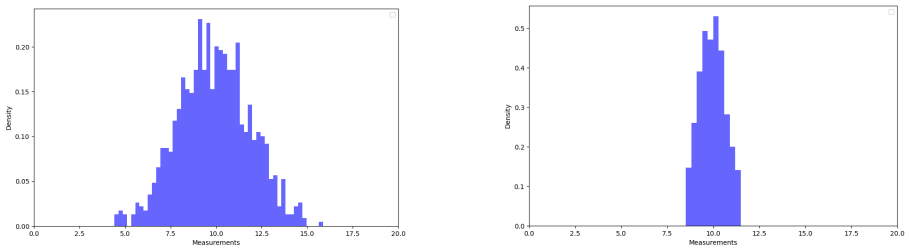
3.1.2 Anomalies vs. outliers

Perhaps the most questionable habit is to use the terms "anomaly" and "outlier" interchangeably. This thesis strongly argues that they should not be used as synonyms, and that while they do overlap to some extent, they are two very different concepts, whose relationship can be summarized as follows:

$$anomalies \approx novelties \neq outliers \approx rare\ items \tag{3.1}$$

Let's start discussing the difference between anomalies and outliers with a simple example: if we measure something multiple times, we know that such measurements should follow a normal distribution. We will have most of measurements close to the true value, some more spread ones, and then some outliers corresponding to the tails of the distribution. Or at least, this is what we *expect*.

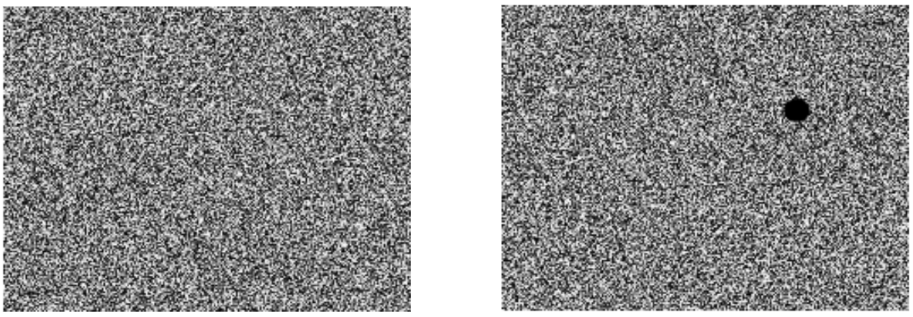
Now, let's assume that one day we find no outliers at all, regardless of how many measurements we perform, and that each measurement always falls extremely close to the previous ones. We would quickly state that there is something anomalous going on, as for example an issue with the instrument or the data processing. In other words, in this case it the *absence* of outliers to indicate an anomaly, since we expected them.



(a) (b)

Figure 3.1: Example "normal" (a) and "anomalous" (b) measurements from an instrument.

Another example where using outliers and anomalies as synonyms fall short is with data where the ordering plays a key role, as series and images. If we look at the distribution of the gray scale color values on the two following white noise images, this will be a uniform distribution with no outliers (given how the images were generated). And one would likely never claim there are outliers in the second image; instead, it is much more probable they would describe it as an anomaly of some kind, probably in the disposition of the pixels. This would be particularly true if the image on the left would be marked as the "normal".



(a) (b)

Figure 3.2: Example "normal" (a) and "anomalous" (b) bi-dimensional white noise.

These two simple examples should be enough to make it clear that while anomalies and outliers are deeply intertwined concepts, they should not be used interchangeably. In other words, *not all outliers are anomalies, and not all anomalies are outliers*. This thesis work will focus on anomalies, and not on outliers.

3.1.3 What is normality, after all?

If we define anomalies as "something unexpected", which is the approach undertaken in this thesis work, we are then implicitly defining anomalies as a complement to normality: what is normal is expected, and what is anomalous it is not. This immediately raises a pretty deep philosophical question: what is normality, after all?

As humans, we perceive our world trough models. From simply throwing a ball and expecting it to follow a given trajectory, to expecting someone not to be mad at us without a reason, or simply for a dice to be fair. Some of such models can be defined in mathematical terms, others in behavioral science ones, and yet other ones using statistics.

When ancient populations saw a solar eclipse, they did not have an explanation for that: it was *unexpected*. So they started to perform various rituals in order to make the sun shine again. Then, we were able to build a model of the solar system, and solar eclipses became much more than expected: they become predictable, and with great confidence.

If we go back to the example of people's height, and we consider an (adult) person 1.3 meters tall, as soon as biology found out that a mutation in the FGFR3 gene caused the so called skeletal dyspepsia, dwarfism become explainable. And even predictable, thanks to DNA testing. Hopefully, no one would call these persons anomalies nowadays (even if they are still outliers in the data). If something, it is the mutation that can still be classified as an anomaly, because its causes are not yet entirely clear. But not its effect, which is instead fully predictable, and thus, expected.

This brings to a very hard truth about defining what is normal and what is not in terms of "expected behavior": if our models could explain anything,

then nothing would be anomalous anymore. We would just "expect" it, since we would have an explanation for everything: our predictions will be always right. Defining anomalies by complementing normality thus rely on the limitations of our models, which we can turn to our own advantage to spot (yet) unexpected behaviors.

However, a crucial part in this processes lie in where to put the boundary between the expected and unexpected: is getting two ones in a row when rolling a dice normal? Probably yes, we all know that it can happen and some board games even have specific rules about it. Can we instead consider it normal to get a series of a thousand ones? Most likely not, and it indicates an anomaly somewhere (regardless whether the dice is loaded or there is a particularly clever throwing mechanism that can control the outcome).

Any anomaly detection technique based on defining normality as the "expected behavior", from simply using an if-then rule to training complex neural networks, require to set such threshold somewhere, somehow.

For example, when looking for anomalies using an if-then rule, the threshold is just a condition on an expected value. If an observation is above (or below) such threshold, then it is considered as anomalous. But how should this threshold value be set, and on which grounds?

When instead looking for anomalies using a model to make predictions about the expected behavior, such threshold is usually the maximum acceptable discrepancy between the predicted and observed values: normality is when the discrepancy is still small, and anomalies start when it gets to big. But what exactly do "still small" and "too big" mean?

Normality is thus a very subjective concept, than can be grounded only if providing a reference rule or model to be considered as such (even in its more abstract form), or a dataset from which to derive it. This can be then used in turn to define the expected, the unexpected, and where the boundary between the two lies.

3.2 Detecting anomalies

Anomaly detection can be approached in a number of ways and using many techniques, which do not necessary depend on the approach chosen. In this context, "approaches" has to be intended as the high-level framing of the problem, regardless of the specific techniques that will be used to implement it, which will be discussed separately (in Section 3.2.2).

3.2.1 Approaches

Three main classes of approaches to anomaly detection have been identified in this thesis: *supervised*, *unsupervised* and *semi-supervised*, depending on how much prior knowledge they assume.

3.2.1.1 Supervised

The supervised approaches, which have to be intended in the broader meaning of the term, address anomaly detection by *already knowing* what to look for. This can assume different aspects based on the various techniques and how they are used.

A first example of supervised anomaly detection are rule-based systems, which can range from simple if-then statements to complex expert systems that can code a variety of cases. In such systems, the notion of normality is predefined based on domain experience. For example, in a combustion engine the maximum expected temperature for the coolant is around 90 degrees Celsius, which if implementing a rule-based system simply translates in marking higher temperatures as anomalous.

In more complex scenarios involving a number of parameters, several conditional statements are chained to cover all the possible combinations. Such systems, known as *expert systems*, have several intrinsic limitations, but due to their nature they are also completely transparent and thus the decision-making process is fully explainable, which can be important in some domains.

In statistics, supervised approaches encompass any technique where a prior threshold is set to differentiate normal and anomalous data instances. For example, setting a cutoff on the data distribution, provided that such distribution contains both normal and anomalous data instances.

Within the machine learning domain, in the supervised learning paradigm a machine learns with input-output pairs: for a given input, the learning process aims at finding a mapping to generate the expected output. In anomaly detection, this requires to provide examples of both normal and anomalous data instances, and therefore to already know how anomalies will look like.

The main advantage of supervised learning over rule-based systems and statistical techniques is not being bound by a set of pre-defined rules written by a human or a fixed algorithm: with the right framework and with enough computing power, any problem that can be framed as a supervised learning problem can be solved, and with great accuracy. However, the risk of supervised learning is to learn too well how to pair the inputs and the outputs, thus resulting in the so called *over-fitting*. This is particularly true when there is not enough data to allow a model to generalize, which is exactly the case in anomaly detection given that anomalies, if known, are rare. Cross validation and other techniques[1] can be used to combat over-fitting and to ensure selecting a model that can generalize well, but always within a given set of input-output pairs: how the model will respond to truly unseen data is completely unknown.

Besides the advantages and disadvantages of the various techniques, a supervised approach to anomaly detection raises a fundamental question: can we still talk about anomalies if we already know them? Or perhaps, assuming for a moment that anomalies always imply bad or dangerous events (which is of course not always the case), shouldn't it be more appropriate to talk about unwanted *states*?

[1] Synthetic data is a new trend being explored in situations of training data scarcity.

This observation is important since it question whether the supervised approach should be considered as capable of performing anomaly detection at all. And again due to anomalies being ill-defined, it is not clear whether the answer to should be a strict "yes", a "no", or if it should be up for discussion.

Many surveys and reviews on anomaly detection make similar considerations on such shortcomings when it comes to the supervised learning approach [20, 63], although usually without clarifying the terminology first, while some of them entirely skip this approach altogether [10, 15].

In any case, if we agree upon Equation 3.1, then the answer is pretty clear: any approach to anomaly detection targeting anomalies which are already known cannot be considered as capable of performing "real" anomaly detection, since it will never be able to spot something "truly" unexpected.

3.2.1.2 Unsupervised

Unsupervised anomaly detection, or in other words anomaly detection without any prior opinions about how the normal and anomalous data instances should look like, is probably what we all ideally dream of. A magic algorithm that, knowing anything about the data, automatically highlights anomalous data points or patterns.

Such approach is however probably more of an utopia than a viable path, and this is because when we think about it, we likely implicitly imagine an advanced artificial intelligence taking the place of a human being, probably ourselves, and in a domain we already know.

But this idea silently ignores that such human *already has* a notion of normality, and can thus easily identify what does not belong to it. Moreover, it risks to be very subjective too: we have all been in a situation where we told someone "I was not expecting this", and the other person, possibly more experienced than us, replied: "well no, this is actually normal".

In statistics, an example of an unsupervised approach corresponds to avoid setting any threshold on the data distribution which contains both normal and potentially anomalous data instances. If a threshold is instead set, an opinion is given, and we immediately fall back in the supervised approach.

Also in unsupervised machine learning, as soon as a threshold is somehow set in order to separate normal and potentially anomalous data instances, then an opinion is given, and the benefits of the unsupervised learning vanishes.

If we instead try to avoid introducing any thresholds or any other prior opinion in order to separate normal and potentially anomalous data instances, then we have no choice other than marking the most "suspicious" data points as anomalous: we simply have no basis to exclude the possibility that such data points are instead "normal".

In other words, in truly unsupervised anomaly detection, in any dataset there is at least one anomaly, by definition. This does not necessarily make this class of approaches useless. They can be great tools for data quickly investigating large amounts of data and pointing a human operator straight to the most "interesting" parts. However, they cannot provide any actionable

insights without a domain expert in the loop validating their results as "real" anomalies, based on an implicit and domain-specific notion of normality.

3.2.1.3 Semi-supervised

This class of approaches does not make any assumptions about anomalies, but makes a strong assumption about what normal data looks like. To this extent, a reference dataset validated as "expected behavior" is used to extrapolate such notion of normality, which is then used to spot any behavior not belonging to it, which is thus anomalous, since "unexpected".

Semi-supervised approaches work exclusively on such reference dataset, which is used not only to capture the "normal" behavior, but also to set the boundary between the expected and the unexpected, thus allowing to ground such elusive concept as discussed in Section 3.1.3.

If we go back now to the example of the car coolant, re-framing it as a semi-supervised anomaly detection approach entails acquiring a dataset of coolant temperature over time, known to be the correct operating mode, then storing the maximum temperature ever reached and marking any *future* temperature above it as anomalous.

Even if very basic, this procedure perfectly match the semi-supervised approach to anomaly detection. The difference between doing exactly the same but in a supervised way lie in how the threshold is set: if it is a human, or if such rule is set automatically based on the data marked as normal.

Having an entire dataset marked as "normal" available opens the door for implementing much more sophisticated techniques to capture the normal behavior rather than just setting a threshold. One could for example start looking at the differences between subsequent data points, correlations between multiple signals, discording sub-sequences, fit a statistical model, or train a neural network capable of capturing complex nonlinear patterns, as will be extensively discussed in the following under various points of view.

Lastly, it has to be noted that in the machine learning realm, the semi-supervised approach to anomaly detection does translates 1:1 to the semi-supervised mode, which is used with different meanings with respect to the use-case. Standard usage refers indeed to the practice of using a small amount of labeled data along with a large amount of unlabeled data to improve model accuracy and generalization, while in anomaly detection based on machine learning it changes the meaning to align with to the same practice described here as semi-supervised.

3.2.2 Techniques

In this section, a number of techniques for performing anomaly detection are discussed, with a primary focus on unsupervised and semi-supervised approaches. While the focus of this thesis work is on anomalies and not on outliers, techniques which target anomalies as outliers have been included as well, for completeness.

Each technique makes use of one or more scoring functions, whose output can be either directly used to provide an estimate of how "anomalous" an observation is, or compressed in a binary fashion (anomalous / not anomalous) using a threshold. How to define the scoring functions and how to set the thresholds (if any) are a crucial part of the anomaly detection process, which is often underestimated and that will be explored further in Chapter 4. For the extent of this overview, only common scoring functions will be reported.

The various techniques are divided in two main classes: *analysis-based*, where the focus is on analyzing the data "as-is", thus looking for data instances that do not fit in the main pattern or distribution; and *model-based*, where a model is used to make predictions about the expected behavior, which are then compared with the observations in order to find any discrepancies.

3.2.2.1 Analysis-based

In their simpler forms, such techniques just look at how the data is distributed and make considerations about it, evaluating the distance between one or more data points and the others in a given distance metric. Such distance can be computed either from a reference value for all the data points (e.g. the mean value) or between relative values (e.g. groups, or clusters, of data points). An example is to look for outliers in the bi-dimensional space consisting of age and height using a given distribution and setting a 3σ threshold.

Extra features can be added to the data points as well, (e.g. the difference with respect to the previous values in sequential data), which allows to accommodate for simple sequential or matrix data to some extent. For example, in a series of temperature measurements, one might be interested not only in the value, but also in how fast it changes, thus considering the first derivative.

Most of these techniques target outlier anomalies, with a few exceptions as the series discords and the one-class SVM techniques. These take a slightly more sophisticated approach than just considering how the data, or some of its features, are distributed: the series discords look at common sequences, while the one-class SVM make use of a higher-dimensional mapping in order to find complex non-linear relationships.

Z-score

The simpler analysis-based technique is to consider the mean value of the data and mark as anomalous any data point "too far" from it. The scoring function is simply the z-score itself (the distance from the mean value), and the threshold defining the concept of "too far" is usually set to three or five standard deviations.

k-Nearest Neighbors (k-NN)

This technique evaluates the distance between a data point and its k-nearest neighbors. The scoring function is the distance itself, which is often cut at a given threshold, based on the distances of the other data points, to differentiate normal and anomalous data points.

Density-based

This technique try to find the best probability density function according to the data distribution, which can be in turn found in a number of ways with probability density estimation techniques. These include choosing the best fitting distribution within a set of candidate distributions, kernel density estimation, discrete histograms, and more.

Once a distribution is selected, the scoring function is the the inverse of the probability, which thus allow to mark data points falling in low probability areas as anomalous.

K-means

This clustering technique tries do divide the space in n centroids. Any data point that does not belong to any centroid is then considered as anomalous.

The scoring function is usually the distance between a data point and the nearest cluster centroids, but it can be also be seen as the dispersion with respect to multiple centroids: if a data point is relatively close to more than one centroids, it can then mean that such data point does not really belong to any of them, and thus with a higher anomaly score than if considering only the distance to the closer centroid. Another option for defining an anomaly score (in unsupervised mode) could be instead to evaluate how many elements each cluster has, in the sense that clusters with few elements can identify a set of (similar) anomalies.

Isolation forest

Also know as *iForest*, this technique tries do divide the data points by cutting the feature space, using random features, and based on the principle that anomalies are harder to isolate with respect to other data points.

The scoring function is the number of cuts required to isolate a data point: the more cuts are required, the more anomalous it is.

One-class SVM

This method tries to find a boundary around the data points mapping the input data into a high-dimensional feature space using a kernel function. Given the usage of kernel functions, it can find complex non-linear relationships within the data, but it only works with semi-supervised approach, since it has to be fitted on normal data instances in order to find the boundary.

The scoring function is the (positive) distance between a data point and the boundary, and the threshold to mark a data point as anomalous is in this case zero (which corresponds to a data point on the border).

Series discords

The series discords technique, originally published in a paper by Keogh in 2006 [45], targets the identification of the most uncommon sequence

in a series. Originally designed to work with an unsupervised approach, it could be extended to work in semi-supervised mode as well.

The scoring function is how much sub sequences differ each other, in a given distance metric.

3.2.2.2 Model-based

Model-based anomaly detection aims at spotting anomalies by considering the discrepancies between the predictions of a given model and the observed values. In this context, a model has to be intended as a generic mechanism of estimating an unknown outcome based on some *input* data (and thus not only predicting in the future, or forecasting). The anomaly score is usually just the difference, in a given distance metric, between the prediction and the observed values, although it can be greatly further evolved and improved.

Model-based anomaly techniques target anomalies as "something unexpected", which, as introduced in Section 3.1.3, is the definition of anomalies adopted in this thesis work. Since outliers can be unexpected too (but not always), this class of techniques can detect them as well, and in a way, it is a superset of analysis-based techniques.

However, unlike analysis-based techniques, model-based anomaly detection works only when there is a mechanism, pattern or dynamic that can be captured and modeled, thus effectively allowing to make predictions. This also implies that the data must be suitable for the task: sequential data, matrix data, or at least data carrying some context is required in order to provide the model with the inputs it needs. Examples of suitable data include words of a text, monthly sales, sensor measurements over time, an image, a 3D point cloud, but also tabular data with more than one dimension.

One of the simpler ways of building a model is probably to just differentiate observational data based on some parameters, which in turn become the input of the model. For example, the average men height globally is around 171 cm, but if parameterized with respect to the of European populations is instead about 178 cm. If we add yet another parameter, as for example how the average height changed over time, we can then ask out model to predict the average height of a man in Europe in the seventeen century. In this case, the model is a simple algorithm which allows to get a prediction based on two input values (the continent and the year). Predicted values can be then compared with observational data, and the mismatches marked as anomalies. For example, in Montenegro the average men height in this century is around 183 cm, and according to such simple model, an anomaly.

It is easy to see model-based anomaly detection and the definition of anomalies as per equation 3.1 coming together smoothly in this simple example. Since the model for predicting the average man height is solely based on the continent and the century, it just can not explain why Montenegro has such a taller male population, which is thus unexpected, or an anomaly. If instead the model would take into account also genetic characteristics and the diet of the

population (the factors believed to cause the higher average man height in Montenegro), then such difference would be expected, and perfectly normal.

Lastly, it has to be noted that the specific type of model used to make the predictions is not relevant when discussing this class of anomaly detection techniques. Instead, it is the type of the model, how it is used, and how the differences between the predicted and the observed values evaluated (and thus which scoring function is applied) that are central to the discussion.

Given the focus of this thesis work on time series data, only regressive models are considered in this section, which are grouped in three main classes: *forecasting models*, *contextual forecasting models*, and *reconstruction models*, which are discussed below.

Forecasting models

The most common technique in model-based anomaly in particular in the time series domain is to use a model to predict future data, or in other words to make a forecast. Such models can work by mapping the entire time domain, and making a prediction for any timestamp (provided that the the further the prediction will be moving away from the data on the time axis, the less accurate it will be), or by using a moving window as the input for the model, as schematically shown in Figure 3.3.

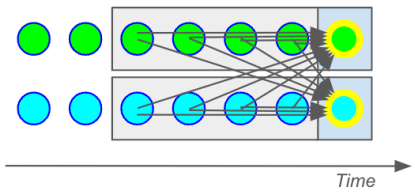


Figure 3.3: Visual representation of a window-based forecaster on a two-variable time series (green dots for the first variable, blue dots for the second). The target of the prediction are both variables (highlighted in yellow).

The forecasting horizon (how many data points in the future to predict) can be set to virtually any value, given that even if the model does not support forecasting the entire horizon in one go, it can always be fed with the predicted data in a recursive way, thus supporting forecasting horizons of arbitrary length[2].

The forecast is the compared with the real observations, and a scoring function applied. This is often just the average error, over the horizon, in a given error metric as the absolute error or the relative absolute error. Thresholds to separate normal and anomalous data points are usually set manually or using simple statistics, as the standard deviation.

If using a forecasting horizon of a single data point, then this class of models models can be applied in an online, streaming fashion, and even

[2]depending on how good the model is, a forecasting horizon too long usually leads to convergence towards the mean value, and is thus not much of use.

in real-time if the circumstances allow for it (e.g. in terms computational requirements). Anomalies can thus be detected as soon as new data points come in.

One of the main advantage of using a forecaster, in particular with respect to the semi-supervised approach, is that going forward along the time axis they will progressively fed with input data already validated as "normal" behavior by the previous runs, thus allowing to keep out-of-sample issues under control.

Example of forecasting models include moving or periodic averages, differential equations, linear regressions, auto-regressive models (as ARIMA, GARCH), numerical models, support vector machines, neural networks, and many more.

Contextual forecasting models

This technique adds some context to the model to help the prediction, or in other words the forecast for a given value is made using other values for the same timestamp, as schematically shown in Figure 3.3.

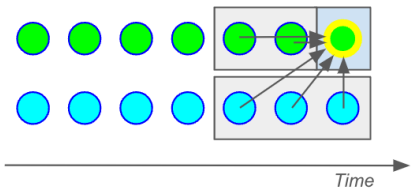


Figure 3.4: Visual representation of a window-based contextual forecaster on a two-variable time series (green dots for the first variable, blue dots for the second). The target of the prediction is in this case just the first variable (highlighted in yellow), which make use of the second variable as well.

While the use-cases for this technique are limited when the goal is the forecast itself, as for example when the target value takes some time before becoming available, it is instead particularly useful for anomaly detection, where the observed target values are always available. The scoring function is the same as for the forecasting models.

One of the drawbacks of such technique, especially in semi-supervised anomaly detection, is to use potentially anomalous data as input for the model. For example, if looking for anomalies in a temperature and humidity recordings, a faulty reading on the temperature might trigger a false positive on the humidity. However, the anomaly would be correctly detected when evaluating the temperature, thus limiting the issue only in terms of ambiguity on where the anomaly actually is. Moreover, as soon as more than two quantities are involved, this issue is less and less relevant.

Example of contextual forecasting models include auto-regressive models with exogenous variables (as ARIMAX), support vector machines, neural

networks, and in general any machine learning or deep learning model where features can be properly set to include the contextual information.

Reconstruction models

Also known as *imputation* or *inpainting* (when referred to images), such models aim to make a reconstruction that fit seamlessly within the surrounding data. The key difference between forecasting and reconstruction lies in the term "surrounding": forecasters predict towards the unknown, while reconstructors can anchor to other data. In sequential data, this can be seen as filing a gap, where the data coming after the gap is known and part of the input of the model as schematically shown in figure 3.5.

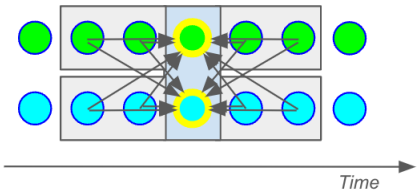


Figure 3.5: Visual representation of a window-based reconstruction model on a two-variable time series (green dots for the first variable, blue dots for the second). The target of the prediction are both variables (highlighted in yellow), using values before and after the gap.

While at first reconstructing gaps instead of forecasting might seem more powerful, it also implicitly assumes the reliability of the surrounding data, which is not always true. In other words, if the surrounding data used as input for the model is also anomalous, the model will either risk to underestimate the anomaly, or worse, provide unpredictable results due to the so called out-of-sample problem.

Moreover, evaluating such models (including model selection) is challenging: the size of the gaps is per-se a parameter, which in real reconstruction tasks should be set by characterizing the data loss of the original (or expected) data. In anomaly detection, given that there are no real data losses to reconstruct but that the data is reconstructed in order to be then compared with the original data, the size of the gap can be chosen freely, although it still has to to be properly tuned and taken into account in the model evaluation.

Examples of reconstruction models include naive models as interpolators and periodic averages, statistical models as ARIMA and GARCH, and deep learning models as the Autoencoders or Generative Adversarial Networks.

3.3 Evaluating anomalies

The last part of this chapter is dedicated to discussing how to evaluate if the anomalies found by a given algorithm are "true" anomalies. Again due to the ill-definition of the concept, this step is not straightforward at all.

A first answer to the question "is this a true anomaly?" could be simply provided by stating that since the definition of anomaly is given by the algorithm itself, anomalies are always "true". This is particularly relevant for the unsupervised approach, because of the observations made in Section 3.2.1.

However, this is course of little help for evaluating the performance of an anomaly detection algorithm, whose goal, as framed in this thesis work, is to detect unexpected behaviors. Perhaps, at the risk of sophistry, the question could be rephrased as: "is this behavior truly unexpected?"

Such question can answered quite drastically if the system being monitored fails as soon as the anomaly is identified: assuming that we (hopefully) did not want this to happen, there is immediate evidence that that anomaly was, indeed, "true".

If instead the system did not fail, candidate anomalies have to be investigated in order to understand if they were part of the normal dynamics of the system (and thus should not have come unexpected, meaning that the algorithm could not capture the notion of normality well) or if it instead they were not (in which case they were true, although not catastrophic, anomalies).

In this context, a set of past known anomalies, spotted either by a human or by the anomaly detection process itself, would seem to be useful in some way. Indeed, while such anomalies will never be enough to approach anomaly detection as a supervised learning problem, as discussed in section 3.2.1, one could decide to use them as a *test* dataset, where to benchmark a given technique.

This can work, but it is dangerous: test data, unlike validation data[3], should be used only once after the supposedly best model for the problem is chosen, as clearly stated by Bishop in his renown book on Pattern Recognition and Machine Learning [9]. To be noted that here, and in this entire section, "model" has to be intended as the anomaly detection process in its entirety, regardless of the technique chosen.

In the most strict approach, test data should be therefore kept secret when selecting, fitting and validating a given model, and uncovered only at the end. And most importantly, the results obtained on the test dataset should be never used to tune, nor to choose, the model.

Given the lack of validation data for anomaly detection problems, the temptation to use the test set to improve the model itself can be high, effectively using it as a validation set which, given its limited size, will inevitably introduce over-fitting (towards known anomalies) and thus potentially backfiring when it comes to detect truly unseen anomalies.

Bishop indeed writes: "*if the model design is iterated many times using a*

[3]Some, including the author, find this way of wording counter-intuitive: it sounds more natural to first test something and then validate it in real-world circumstances. However, to avoid confusion, this thesis will stick with the widely accepted wording used by Bishop.

limited size dataset, then some over-fitting to the validation data can occur and so it may be necessary to keep aside a third[4] *test set on which the performance of the selected model is finally evaluated.*"

Along similar lines, in his famous book "The Signal and the Noise", Nate Silver makes a pretty clear point when it comes to evaluate models used to predict earthquakes on past data: *"it is crucial to keep 'retrodictions' and predictions separate; predicting the past is an oxymoron and obviously should not be counted among success".*

This is not the entire story though, otherwise fields as machine learning and deep learning would not exists at all, given that they are entirely based on evaluating models on what Nate Silver calls "retrodictions". But there is a key difference: when a machine learning model is evaluated on the past, there is enough data to ensure that "retordictions" are statistically reliable, not over-fitted, and that can thus be used as predictions.

When it comes to detecting earthquakes, this assumption is just wrong: there is not enough data to check whether a given model can correctly capture the signal over the noise in a statistically significant way and without over-fitting it, and that can thus generalize enough to allow relying on its "retordictions". It's no coincidence that the chapter where Nate Silver wrote the quote above is titled "desperately seeking signal", and in anomaly detection it is exactly the same: there is just not enough data.

In any case, even if correctly using such small set of known anomalies as a test set, or in other words without using it by any means to improve the accuracy of the anomaly detection process, it would never be representative for truly unexpected behaviors, and thus lead to misleading accuracy metrics.

Therefore, if approaching anomaly detection with genuine intellectual curiosity, then a hard truth must be faced: *anomaly detection techniques just cannot be effectively evaluated.*

[4]For Bishop, in this context, the first set he is implicitly referring to is the validation set, and the second one the test set.

Chapter 4

Proposed methodological framework

4.1 Design choices

In order to address the challenges outlined in Section 2.4.2, and the observations of the previous chapter, the methodological framework proposed in this thesis work makes precise design choices.

These choices are based on optimizing the methodology for the various requirements when evaluated as a whole, and for real-world scenarios, meaning that are not inherently the best ones when evaluated as single entities.

Moreover, also the tradeoffs made are not necessarily the best possible ones, given the huge amount of work required to properly optimize all of them given the various constrains, and leaves room for improvement.

4.1.1 Data processing

Before discussing the data processing, it is worth mentioning also the data acquisition step. While not necessarily in control when implementing an anomaly detection process, there are still some important requirements on data acquisition that should be fulfilled when possible.

The first and foremost of them is to acquire data with UTC (or epoch) timestamps. Storing the timestamps in local format will inevitably cause issues over Daylight Saving Time (DST) changes, and either cause to get duplicates (when the clocks are moved backward) or gaps (when clocks are moved forward. Data should also be never sent, or stored, in an unordered way. This can indeed cause to feed the data processing system with newer data without any awareness that past data has still to be waited for. Also, when possible, data acquisition should be performed in a synchronized way, e.g. not at generic 60 seconds intervals from the boot of the data acquisition device, but at the same moment across all the data acquisition devices.

In case of a failure when reading a sensor, no placeholder numerical values

should be used (as zero or -99), but either no data should be generated, or a null value used. Lastly, in case of a setting where the sampling rate can change over time, it has to be kept track of each sampling rate change, either as metadata of the data itself or separately, but it should be always possible to reconstruct which sampling rate was in effect when.

Moving to data processing, the first step is the processioning, which usually involves resampling the data (unless it is acquired at synchronized time steps, as mentioned in the previous paragraph). Assuming that the design choices about the data acquisition are usually not in control by who then implements the anomaly detection, this step has also the role to fix all the potential issues. It can include a buffer to handle unordered data, heuristics to detect sampling rate changes, translating time stamps from local times to UTC and handling the intrinsic ambiguity when the DST changes backward to prevent duplicates, as well as computing the data loss. The data loss in particular is of paramount importance for anomaly detection, as it prevent to fit or apply a given technique on potential artifacts.

The resampling step, when generating the sample for the time t, must wait for a sample coming after (or exactly at) the the timestamp $t + 1$. There is no way to remove such lag (unless acquiring data in a synchronised way): the resampler has just not enough visibility on the forthcoming data to generate a value for the timestamp t, let alone to mark as lost the sample. When generating the value for the timestamp t, the data loss has to be computed according to the underlying data. Intuitively speaking, a sample generated right before a gap in the data will have a data loss of about 50% (given than the only reliable data points are on its left), while samples in the middle will have a data loss of 100% (meaning that they are fully interpolated). The data loss must be stored together with the values of the data points, or allow for quick and simple lookup.

Lastly, any parameter estimation or inference must be paused when data losses are detected, according to a thresholding mechanism. This is particularly relevant when fitting a predictive model or when computing the anomaly detection parameters, given that it could undermine the entire anomaly detection process, but also when performing the anomaly detection itself, as it can easily generate false positives due to the garbage-in garbage-out paradigm.

4.1.2 Anomaly detection

As per Section 3.2.1.1, a supervised approach to anomaly detection would introduce too much rigidity, and most importantly to already know at least some of the anomalies, which in particular with respect to newly built or newly monitored critical systems it is rarely the case.

An unsupervised approach would instead trigger many alerts, since in such mode it is not possible to set any threshold, and require a human in the loop as discussed in Section 3.2.1.2. In particular with respect to critical systems, where anomalies can trigger potentially urgent alerts in an automated way in order to preserve the operations, this is not a viable approach.

The choice was thus to employ the semi-supervised approach, which is also particularly suitable for critical systems because of the commissioning period, where the system is closely monitored by domain experts in order to ensure it operates smoothly. Such period can naturally generate a reference dataset for the "normal" operating mode, given that in case there were any issues or anomalies these are usually promptly spotted and reported and thus easy to exclude form such dataset. Moreover, the semi-supervised approach is the only one that allows to remove most of the ambiguity about the definition of anomalies.

The technique chosen to perform the anomaly detection is the model-based technique. This choice was made, together with the semi-supervised approach, for a number of reasons.

First of all, when it comes to time series data, models are naturally suitable for capturing patterns, which with other, analysis-based techniques are harder to take into account. One of the few exceptions is the series discords technique, which can find the most unusual sub-sequences in sequential data. While this technique was originally designed to work with an unsupervised approach in mind, and for univariate time series, it is possible to extend it to make it work in semi-supervised mode and for multivariate time series. However, it does not allow to work without a lag time, as it requires for a sequence to be complete before it can be compared with the other ones. It also requires to compare any new sequence with all the others, a procedure which, even if the authors prove that can be greatly optimized with respect to the brute force approach, can still be computationally very expensive, in particular when working in a streaming and online setting.

In second place, unlike other anomaly detection techniques, using a model allows to incorporate valuable prior domain knowledge, and most importantly, to obtain an indirect (yet precious) evaluation of the anomaly detection effectiveness. Intuitively speaking, a model capturing well the dynamic of the underlying system will also translate in more effective anomaly detection, while a model performing poorly will not provide particularly good results.

Lastly, evaluating the error distribution of a model in "normal" conditions allows to make probabilistic considerations about it, and thus to assert in a more confident way whether a new observation is compatible with such model (and thus "normal") or not (and thus "anomalous").

With respect to the class of models to use, forecasting (or contextual forecasting) models are a natural choice given the requirement of evaluating new data in an online and streaming fashion. However, there is also another and more subtle argument in support of choosing forecasting models over other ones. If a model is fed with data other than the from the past (which has thus been already evaluated for anomalies), then there is no guarantee that such data would not be already anomalous, which can in turn cause to either underestimate the anomalies, or worse, to get get unpredictable results due to the out-of-sample issue. When designing a framework for robust anomaly detection, this is an important advantage.

Within the various forecasting models, the choice is to use those relying only on a portion of the data to make predictions, or in other words using a moving window. This is particularly important because it allows the same model to be used repeatedly on the incoming data without any retraining or updating. Constantly retraining or updating a model is indeed not a viable path not only because of the heavy computational and input/output activity, but also because when working in a semi-supervised approach it would be counter-productive: a model constantly adapting to the new observations would progressively deviate from the validated notion of normality, which is exactly what has to be avoided.

With respect to the forecasting horizon, the choice is to work with single step-ahead predictions. This is mainly because anomalies are required to be detected as soon as possible, but there are other reasons including to prevent the propagation of errors and to allow a simpler modeling of the error distribution. Moreover, there is no evidence that a multi step-ahead prediction would work better in the context of anomaly detection. In any case, when a longer lag time can be accepted, and a multi step-ahead prediction proved to be more suitable, the framework proposed here can be extended to compare predicted and observed sequences rather than just punctual predictions and observations, and it is part of the themes envisioned as future work.

The scoring of anomalies must be simple and intuitive, and allow for clearly assessing the degree of suspicion of a potential anomaly. Most research works do not focus on this aspect, and for model-base anomaly detection a typical choice is to just use the error between the predicted and observed values, which carry little information about what a given score actually means. Moreover, it does not allow to compare indications from anomaly scores generated by different models, which is instead important when developing a robust methodology since it allows working with ensemble models, that can greatly help in improving the overall anomaly detection capabilities.

In order to better quantify the degree of suspicion for a new observation of being anomalous, and to allow comparing the outputs from different models, this thesis work introduces a new anomaly score, called the *anomaly index*. Such index, which will be introduced in the next section, is based on the probability of a new observation to be compatible with a given model of normality, and can range from zero (not an anomaly) to one (almost surely an anomaly).

How to set such boundaries is a key step that will be covered in the following section as well, partly answering the question Irwin posed back in 1925: while it is true that how to set $P(\lambda)$, or any other threshold, to differentiate between normal and anomalous data instances is a *"matter of opinion that the mathematician cannot settle"*, it is also true that in semi-supervised anomaly detection there is an important difference: we are cheating, or in other words we already know what normality is supposed to look like.

4.2 The anomaly index

Given how challenging it is to quantify the degree of suspicion for a new observation to be anomalous, a considerable effort in this thesis work has been spent on this topic. While most research work focus indeed on improving the anomaly detection techniques, which as argued in section 3.3 can also risk to translate in a task just serving its own sake, little work has been carried out on how to use, nor present, their results.

Usually, anomaly scores are just distance metrics: from a neighbour, from a cluster, from a boundary, or between the predicted and observed values. However, this approach has two main limitations: it makes it hard to interpret the scoring, and it does not allow to compare the outcome of different models.

For this reasons, this thesis work proposes a new anomaly score for model-based anomaly detection, based on how much a given observation can be considered as incompatible with a given model of normality.

Such score was designed to overcome the two limitations mentioned above, providing a clear interpretation while ranging from zero to one in order to allow for an easy comparison, and named *anomaly index*.

4.2.1 Formal definition

In model-based anomaly detection, given a predicted value p and an observed value o, one could immediately define an anomaly score s by just computing the distance between the predicted and actual value:

$$s_1 = |p - o| \tag{4.1}$$

Depending on the use case, a slight improvement is to consider the magnitude of these values with respect to the data, typically using a relative error form:

$$s_2 = \left| \frac{(p - o)}{o} \right| \tag{4.2}$$

In general, given an error metric E, this could just be used as-is to define an anomaly score:

$$s = E(p, o) \tag{4.3}$$

However, defining a score in such way, which is a common choice for most of the works found in literature, has to main issues:

1. it has no probabilistic interpretation nor meaning, and
2. it does not allow to compare anomaly indications originating by different models, either on the same or different datasets.

The first point is of paramount importance when anomalies must trigger actions in the real world, because as already introduced in section 2.4.2 a scoring

method that cannot provide a clear indication about the degree of suspicion of potential anomalies is of little practical use.

The second is instead simply essential for any use case involving more than a single univariate time series, let alone using ensemble models: comparing different anomaly indications is a key step for any anomaly detection process dealing with anything slightly more complex than toy problems.

To overcome such limitations, we can consider the error distribution and use it to evaluate how likely it is for a given error between a predicted value p and and observation o to occur. More in detail, we consider:

- a dataset D with data items $o_1, o_2, o_3, ..., o_n$, also refereed to as the observed values;
- a model M capable of making a prediction p for an observed value o, and in particular making predictions $p_1, p_2, p_3, ..., p_n$ for each o_i in D;
- an error metric E to compute the error $e(o, p)$ between predicted and observed values, applied to all the pairs o_i and p_i; and
- a unimodal distribution function $f(e)$ fitted on all the prediction errors $e(o_i, p_i)$, as a normal or generalized normal distribution (Figure 4.1).

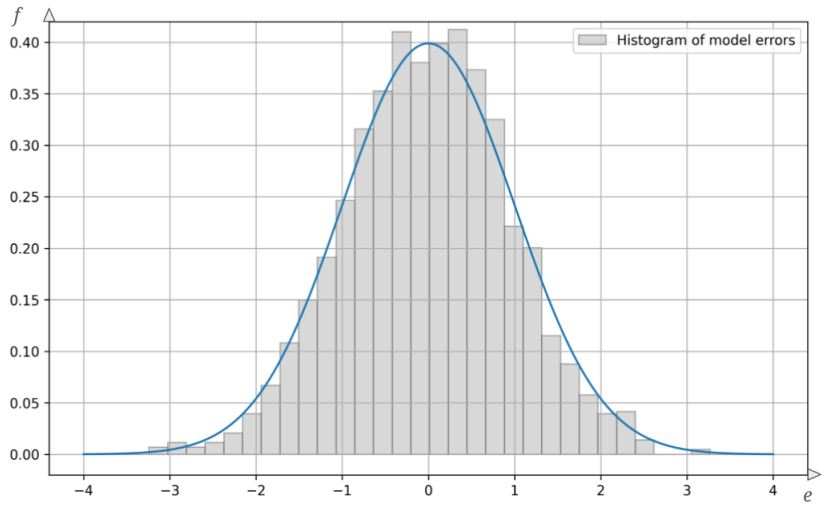


Figure 4.1: An example (normal) distribution function f fitted on the errors of a given model.

We then define the *adherence* of a generic, and potentially new, observed value o to the model M with respect to the dataset D as:

$$h(o, p) = \frac{f(e(o, p))}{max(f)} \tag{4.4}$$

where p is the prediction the model made for the value o. Or simply:

$$h(e) = \frac{f(e)}{max(f)} \tag{4.5}$$

The values of such quantity span the 0-1 range given how it is defined (Figure 4.2), and if we consider its complement to one $(1 - h)$, we obtain the inverse concept: an indication of how much an observation o is *not* adherent to our model M, with respect to the dataset D.

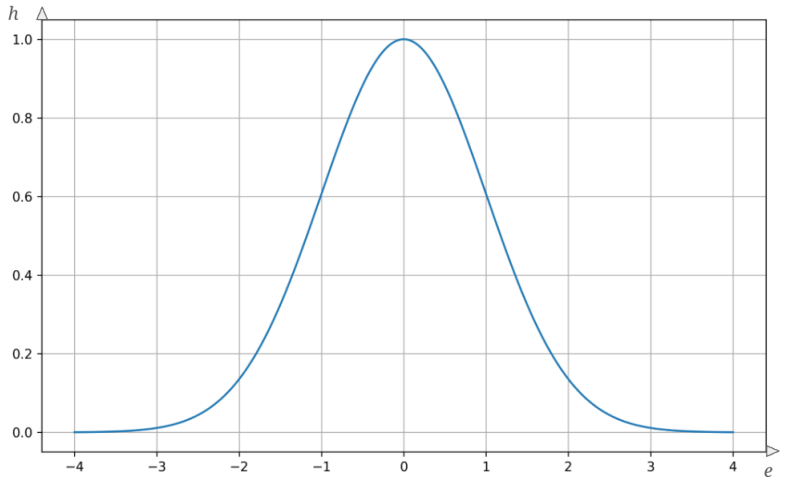


Figure 4.2: The adherence defined according to equation 4.5 plotted with respect to the example (normal) distribution of Figure 4.1.

At this point, one could be tempted to assign a meaning in terms of abnormality to the quantity $1 - h$ as-is: the lower the adherence to the model gets, the more we suspect an anomaly.

While on the right path, this is not particularly useful for two reasons. First, we cannot declare a suspicion of anomaly as soon as an observation starts deviating from the model, which has an intrinsic error that cannot be neglected: the above mentioned quantity would indeed assign a suspicion of anomaly greater than zero to any observation, even when the error is very low. In particular with respect to the semi-supervised approach, where the maximum error committed by the model on the normal data is still to be considered as perfectly normal, this is not a viable choice. Second, given that this quantity would approach 1 asymptotically, but never reach it, it would never allow to state if a given observation is to be considered as not adherent with the model, and thus an anomaly, in a clear way.

Instead, we need *boundaries*: where to start getting suspicious about a given error (and consequentially the observation that generated it), and where to stop being just suspicious and instead assert that we are (almost) certainly we are facing an anomaly. How to set such boundaries will be discussed in the next section; for the time being, let's just assume that they are set somehow, and let's name them h_{start} and h_{end}. Figure 4.3 shows a visual representation of this concept, together with the errors associated with the respective adherence boundaries.

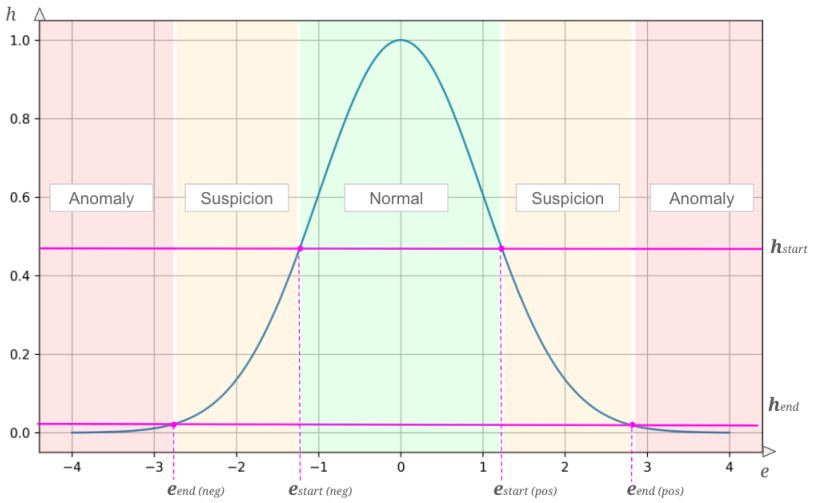


Figure 4.3: Visual representation of the adherence boundaries h_{start} and h_{end} and the associated errors.

Once the lower and upper boundaries h_{start} and h_{end} are identified, we can proceed by rescaling the adherence of a given observation with respect to the adherence at such boundaries, using the following formula (which rescales the adherence values within the boundaries in the 0-1 range):

$$\tilde{h}(e) = \frac{h(e) - h_{end}}{h_{start} - h_{end}} \tag{4.6}$$

As before, we can consider its complement to one $1 - \tilde{h}$, which still represents how much an observation o does not adhere to the model M with respect to the dataset D, but within the adherence boundaries we set.

We can therefore proceed in formulating the overall expression for a possible anomaly index j based on such considerations:

$$j(h) = \begin{cases} 0 & \text{for } h \leq h_{start} \\ 1 - \tilde{h} & \text{for } h_{start} < h < h_{end} \\ 1 & \text{for } h \geq h_{end} \end{cases} \tag{4.7}$$

However, an index defined in this way would do a poor job in differentiating anomalies based on their order of magnitude, which is important when evaluating them and deciding how to react. The boundary values in the example of Figure 4.3 are indeed quite fictional, and a more realistic scenario would look more something like Figure 4.4.

A logarithmic scale would be therefore much more suitable, as it can bee seen, again visually, in figure 4.5.

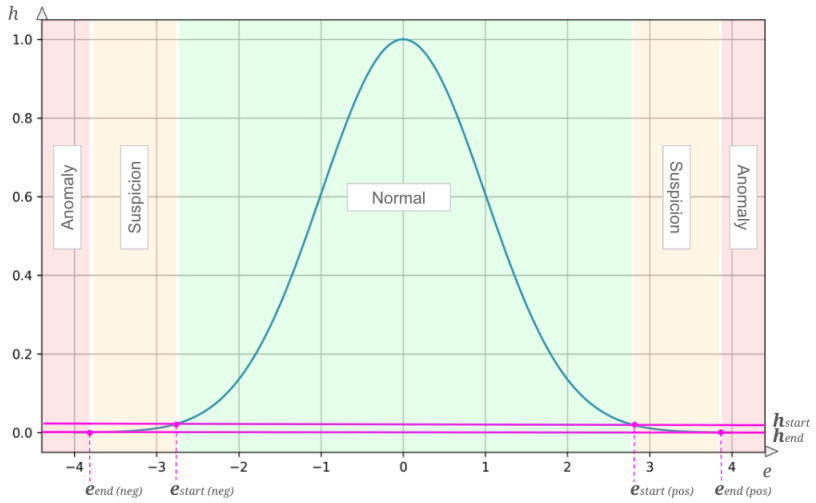


Figure 4.4: Visual representation of the adherence boundaries h_{start} and h_{end} and the associated errors in a more realistic scenario than Figure 4.3.

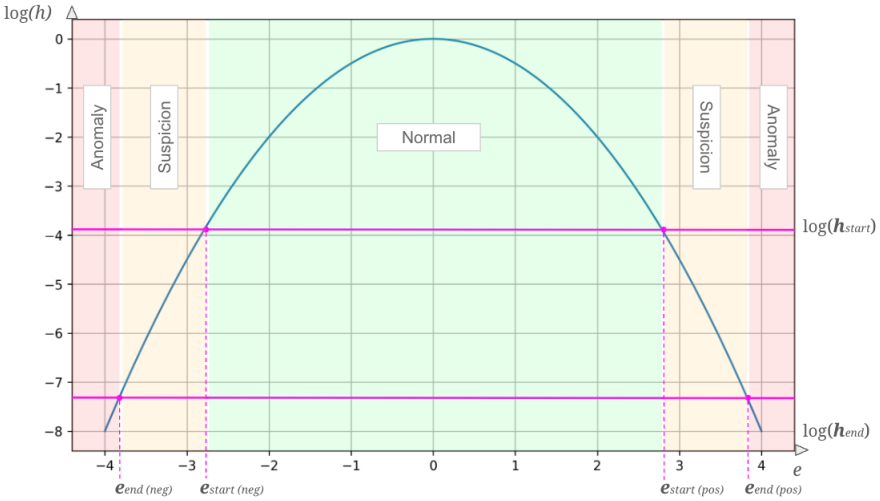


Figure 4.5: Visual representation of the adherence boundaries h_{start} and h_{end} and the associated errors on a logarithmic scale.

With a few trivial considerations, it is easy to see that the following equation holds true:

$$1 - \tilde{h} = 1 - \frac{h - h_{end}}{h_{start} - h_{end}} = \frac{h - h_{start}}{h_{end} - h_{start}} \tag{4.8}$$

This allows to rewrite the expression 4.7 as:

$$j(h) = \begin{cases} 0 & \text{for } h \leq h_{start} \\ \dfrac{h - h_{start}}{h_{end} - h_{start}} & \text{for } h_{start} < h < h_{end} \\ 1 & \text{for } h \geq h_{end} \end{cases} \tag{4.9}$$

On such expression, it is easy to apply a logarithmic transformation, which leads to the final form of the anomaly index i:

$$i(h) = \begin{cases} 0 & \text{for } h \leq h_{start} \\ \dfrac{log(h) - log(h_{start})}{log(h_{end}) - log(h_{start})} & \text{for } h_{start} < h < h_{end} \\ 1 & \text{for } h \geq h_{end} \end{cases} \tag{4.10}$$

It can also be noted that in this expression all the adherences h can be replaced with just the error distribution function f, given that the dividend of equation 4.5 cancels out, thus allowing to express the anomaly index as a function of the error (and its distribution) only:

$$i(e) = \begin{cases} 0 & \text{for } e \leq e_{start} \\ \dfrac{log(f(e)) - log(f(e_{start}))}{log(f(e_{end})) - log(f(e_{start}))} & \text{for } e_{start} < e < e_{end} \\ 1 & \text{for } e \geq e_{end} \end{cases} \tag{4.11}$$

where $e = E(o, p)$, and e_{start} and e_{end} are the boundaries in terms of the error, to be set either directly or obtained by reversing equation 4.5.

This makes the anomaly index particularly convenient to manipulate and implement, although defining the boundaries in terms of the error requires assuming a symmetric distribution, or to split the cases between the right and left parts of the distribution.

Lastly, in case of models predicting multiple quantities, one could extend this reasoning to the n-dimensional case; or simply handle it with separated indexes.

4.2.2 Setting boundaries

How to set the boundaries of the anomaly index is a key step, given that they represent where to start getting suspicious about a prediction error (and consequentially the observation that generated it), and where to stop being just suspicious and instead assert that there is (almost) certainly an anomaly.

In the unsupervised approach, such boundaries does not leave much room for discussion. As already explained in section 3.2.1, in this approach the only reasonable choice is to consider the most "isolated" data point as surely

anomalous, which in model-based anomaly detection translates in the bigger prediction error. The upper boundary would then be set to such value, while the lower one to any value where the suspicion is considered to be worth investigating, as for example when committing a prediction error above a three sigma cutoff.

In the semi-supervised approach instead, errors up to the maximum error committed by the model on the normal data have to be considered as still perfectly normal. Above such error, our suspicion about observing an error generated by another mechanism rather than by an extreme manifestation start to arise, and keep arising up to a point where we stop just being suspicious and where we assert that such error is (almost) surely due to another mechanism, or in other words, an anomaly.

Such threshold can be set in many ways depending on many factors, and it is part of what a mathematician cannot settle, citing again Irwin. Some of these ways include:

- using a somewhat intuitive yet opinionated value (e.g. the double of the maximum error);
- using an extremely low value (e.g. $1/10^{20}$, $1/10^{30}$);
- making some considerations about the maximum rate of false positives that can be tolerated, in a backward manner.

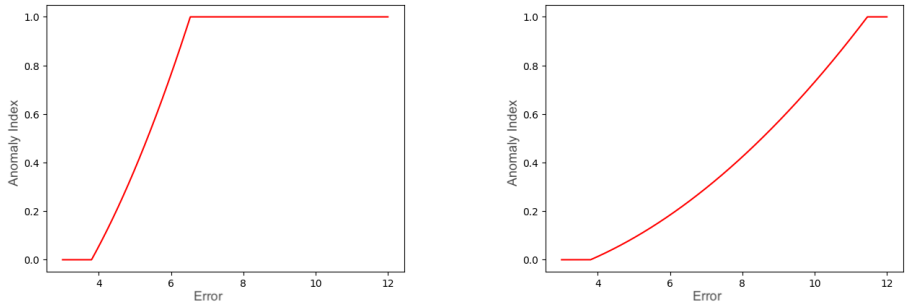
The plots if Figure 4.6 show how the anomaly index looks like in an example case, using two different upper boundaries ($1/10^{10}$ and $1/10^{30}$). If using the first, an error twice as the maximum error committed by the model on the normal dataset would be marked with an anomaly index of 1, while if using the second, it would be marked with an anomaly index of about 0.4.

It usually should be more desirable to set a very low probability boundary (as the in the second case) for the upper value, and rather put a threshold on the index itself to decide whether to fire an alarm or not, so that the magnitude of anomalies are compressed only when extremely unlikely.

It can also be useful to use the few known anomalies, or artificially created anomalies, to tune such boundary, which is perfectly legit. One could for example add to the data an anomaly corresponding to a complete failure (e.g. a value dropping near zero) and check whether the chosen upper boundary would mark it with a reasonable anomaly index.

It is also interesting to see how powerful it is to rely on a given notion of normality when it comes to fit an error distribution. While it is true that the famous threshold the mathematician cannot settle is always something to deal with, it is also true that in the semi-supervised approach to anomaly detection we are using a precious hint: we know how normality looks like.

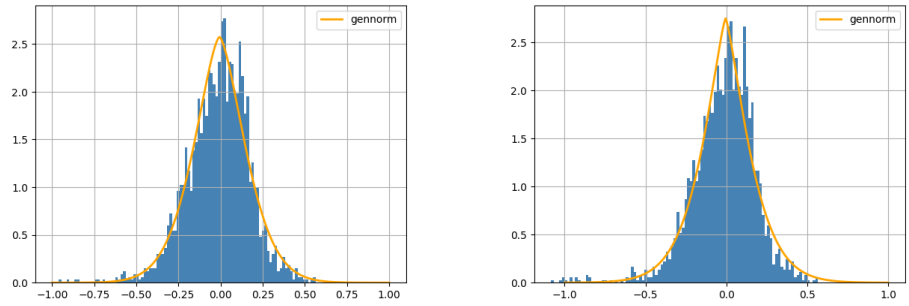
The following two error distributions were fit form the prediction errors of a simple linear regression model, which was in turn fitted on almost the same dataset (a simple univariate time series of air temperature). Such datasets, about two months long, differed for just two days: in the first dataset there



(a) (b)

Figure 4.6: The anomaly index plotted for an example model and dataset, using as lower boundary the maximum error (3.8) and as upper boundary a probability to obtain an error greater than the boundary of 1 in 10^{10} (a) and of 1 in 10^{30} (b).

was only data acquired in the normal operational mode, while in the second there was an anomaly (whose characteristics are now not relevant).



(a) (b)

Figure 4.7: Error distributions on just "normal" data (a) and on both "normal" and "anomalous" data (b).

Some of their parameters change significantly, and the p-value changes drastically between the two, as reported in table 4.1. And it makes perfect sense, given that the error introduced by such anomaly is from *another* distribution.

	beta	loc	scale	p-value
Distribution a	1.55	-0.0067	0.21	0.112
Distribution b	1.31	-0.0051	0.19	0.008

Table 4.1: Parameters of a generalised normal error distribution fitted on the prediction errors of a model fitted on just "normal" data (distribution a) and on both "normal" and "anomalous" data (distribution b).

4.2.3 Error metrics

In the previous section, we used a generic error metric $E(p, o)$. However, the error metric to be chosen for the anomaly detection process is a key element, given that any subsequent consideration stems from its choice.

Many error metrics exists to chose from, and in literature there are various attempts to define more and more meaningful error metrics.

Error metrics can be either scale-dependent or scale-independent, subject to whether they evaluate the differences between the predicted and actual values in a way that is dependent to the scale of values involved, or not. Usually, scale-independent error metrics are also referred to as just "relative". The most common error metrics are briefly summarized in the following.

Error (E): the simpler error metric, which computes the signed distance between a predicted and observed value, in a scale-dependent way:

$$E(o, p) = o - p \tag{4.12}$$

Squared Error (SE): a square operation is applied on the difference between the observed and predicted values:

$$SE(o, p) = (o - p)^2 \tag{4.13}$$

Logarithmic Error (LE) in this case a logarithmic transformation is applied bot to the predicted and observed values:

$$LE(o, p) = log(o) - log(p) \tag{4.14}$$

Relative Error (RE): it divides the difference between the observed and predicted values by the observed (or predicted) value:

$$RE(o, p) = \frac{o - p}{o} \tag{4.15}$$

Percentage Error (PE): this is just the relative error multiplied by a factor of 100, and it is common to refer to relative or percentage error metrics interchangeably. Formally, it is defined as:

$$PE(o, p) = \frac{o - p}{o} \cdot 100 \tag{4.16}$$

Each of the above metrics can be also defined in their absolute counterparts to obtain only positive values (except for the SE which is already positive-defined), by just applying the absolute value operation and pre-pending an "A" (for Absolute) in their acronyms: AE, ALE, ARE, APE. In such case, the anomaly index defined in section 4.2.1 it is required to consider only the positive side of the domain.

In order to get instead an error metric capable of taking into account multiple observer-predicted pairs, usually an average operation is added on positive-defined metrics, which is customarily denoted with the letter M. So the APE

becomes the MAPE (Mean Absolute Percentage Error), the AE becomes the MAE, and the SE becomes the MSE (Mean Squared Error).

Other operations could then be applied, as for example the square root, which if applied on the MSE brings to the RMSE (Root Mean Squared Error). The focus within this section is however on *punctual* error metrics instead aggregated ones, so these are no further explored.

The choice of which metric to use depends on the dataset and the context, the first decision key being whether adopting a scale-dependent error metric or not. For example, in an energy grid it is likely better to use a error metric, given that bigger errors are symptoms of bigger amounts of energy involved, and thus of more substantial consequences. In monitoring the temperature of a machine instead, there ins nothing inherently worse in making an error when the temperature is high than when it is low (again depending on the context).

Other considerations can involve the definition domain, stability, and fair weighting. For example, scale-independent (or relative) error metrics that perform a division by either the observed or predicted values are not defined when the dividend is zero, which might not suitable for many use cases (e.g. a temperature that can be either positive or negative). Similarly, such metrics are unstable around a zero value of the dividend, given that little fluctuations translates to extremely big ratios.

With respect to weighting instead, a good error metric should equally penalize under-estimations and over-estimations, while most of scale-independent error metrics as the ARE and APE does not. For example, in case of over-estimating the observed value by a factor of ten, the APE has a value of 900%, while in case of under-estimating it by the same factor, the APE has a value of just 90%.

Some attempts have been carried out in order to overcome such issues, however they usually just lead to complicating things [31] without providing tangible benefits. While such approaches tried to define a generic go-to default indeed, in real-world circumstances the choice of the error metric ultimately solely depends on the dataset and the context.

4.3 Candidate models

This section describes a set of forecasting models that can be mixed and matched for model-based anomaly detection within the framework proposed in this thesis, together with their advantages and disadvantages.

It has to be noted that some well known models for time series forecasting are not suitable and thus have not been included in this set. For example, most statistical methods such as ARIMA and its derivatives[1], GARCH, Exponential Smoothing, Facebook Prophet [81], etc. cannot be applied in an online fashion: they require continuity of the data to be forecasted with respect to the fit data, meaning that accounting for new data requires to re-fit the model on the entire dataset, which is just unfeasible.

[1]Such as SARIMA, ARIMAX, and VARIMA.

More classical numerical models are also not included, although might be technically suitable. However, such models are usually domain-specific, require a-priori calibration and are computationally heavy.

The forecasting models presented in this section are classified in four main categories, based on their complexity, the assumptions they make about the underlying data and their abstraction capabilities. Such categories are: naive, statistical, machine learning and deep learning.

4.3.1 Naive

Naive method are, as the name suggest, extremely simple methodologies often used as benchmark or when there are severe constrains on the computing resources.

Feed forward The simpler technique to perform a forecast is just to assume that the latest value is the best approximation for the future one. While of limited predictive power, it can set a solid baseline for evaluating other forecasting methods, and in the context of anomaly detection can easily spot unexpected spikes or discontinuities at nearly no computational cost.

Moving average This method takes the average of the values over a window of n item in order to forecast the next value. It smooths out short-term fluctuations and highlights longer-term trends. Also inexpensive in terms of computational power, in the context of anomaly detection it can be useful to detect increasing noise levels or faster changes than usual in the dynamic of a signal.

4.3.2 Statistical

Statistical methods are more complex than the naive, being built on probability theory and hypothesis testing, usually in order to understanding relationships between variables.

Periodic average This technique requires a periodic signal, and it simply computes the average value for each part of the period (e.g. hourly the average temperature for each hour of the day). The average value can be used as-is, weighted with respect to the contextual conditions, as the offset of the previous n periodic averages. Such a simple model can make surprisingly accurate predictions on simple periodic signals (as the air temperature), which together with its extremely low computational footprint it makes it a great model for experimenting.

Seasonal Decomposition This class of techniques evolves further the concept of a periodic average by taking into account more complex dynamics, or in other words to model more than just a single periodicity. Such techniques are commonly used to account for (as the name suggest) seasonal changes in the trend (e.g. gas consumption in winter vs in summer).

4.3.3 Machine learning

Machine Learning (ML) methods aim at learning how to optimize a given function (usually called the *loss*) based on the data, and do not make any assumptions on the underlying structure (unlike statistical and naive method). However, they still make assumptions about the features of such data.

Linear regression This method models the relationship between a dependent variable and one or more independent variables, and in particular it learns a linear relationship from historical data that best fits the patterns. Predictions are then obtained by extending the learned relationship in the future (or by providing as input the future values of the independent variables).

SVM Support Vector Machines (SVM) are used for forecasting by fitting a hyperplane that minimizes the prediction error. It's particularly useful for capturing non-linear relationships in time series, but can be computationally intensive and sensitive to parameter tuning.

GBM Gradient Boosting Machines (GBM) and other boosting methods like GBM, XGBoost, LightGBM, and CatBoost are highly effective for forecasting, as they iteratively train trees to reduce errors. They excel with non-linear, complex relationships and are quite popular in time series forecasting applications.

4.3.4 Deep learning

Deep Learning (DL) models do not make assumption about the features on the underlying data, unlike ML models, and are the most flexible solutions. DL are a subset od ML, given that they are also based on learning how to optimize a loss function. They are mainly based on Artificial Neural Networks (ANNs), or just Neural Networks, (NNs), which mimic brain neurons.

Many neural network architectures exist for modeling time series data. The main ones can be divided in five classes, based on the state of the art: RNN, LSTM, GRU, TCN and Transformer-based, which are summarized in the following list.

RNN Recurrent Neural Networks (RNNs) are particularly useful for sequence data, such as time series, because they retain a memory of previous inputs, allowing them to capture temporal patterns. While not explicitly designed for long-term dependencies, they can handle certain temporal dynamics effectively.

LSTM. Long Short-Term Memory (LSTM) neural networks are a specialized version of RNNs that make use of an internal memory cell structure. This allows them to effectively remember relevant information over longer time intervals, making them better suited for capturing long-term dependencies in time series data.

GRU. Gated Recurrent Unit (GRU) neural networks are similar to LSTMs but use fewer gates, making them simpler and faster to train while still handling long-term dependencies well.

TCN. Temporal Convolutional (TCN) neural networks make use of causal convolutions to process time series data in a parallel fashion. Thanks to this approach, they do not suffer from the same vanishing gradient issues as RNNs and LSTMs, and are more efficient.

Transformer-based. First introduced in 2017, Transformer-based neural networks [85] make use of a self-attention mechanism that allows them to capture relationships between any time step, and not only in a linear way as all the above architectures. This ability to model long-range dependencies without relying on sequential processing makes them particularly well-suited for time series data.

4.4 Model selection

4.4.1 General considerations

Selecting the best model for a given task is a wide topic which involves several use-case dependent variables. The first and foremost is the accuracy, but also the performance (to be intended as the required computational resources) can be a driving factor towards choosing a given model over another. Other factors can include stability with respect to out of sample data, contained maximum errors, explainability, and so on.

With respect to model-based anomaly detection, the two main factors to consider beyond the average accuracy are the maximum error and the goodness of the fit of its distribution, since in the approach proposed in this thesis work it will be used to draw considerations about new observations.

Moreover, in particular in the domain of critical systems, performance is somewhat secondary, given the high stakes involved.

4.4.2 Fitness function

When evaluating a candidate forecasting model for model-based anomaly detection, and in particular when making considerations about its error, there are three main quantities to keep under control:

1. the *aggregated error*, which provides an indication of the quality of the model on average;

2. the *maximum error*, given that we are more interested in models that make a limited error in the worse cases, even if less accurate on average, rather than models that are very accurate on average but that can make potentially very large errors; and

3. the goodness of the fit of the error distribution, given that if the error cannot be modeled then it cannot be used to drawn conclusion (as the anomaly index does).

In order to combine such requirements, a common strategy is to define a *fitness function* to describe how fit is a given model for a given task, that can be in turn used to assess how suitable it is for the task, using a single metric.

Any of the metrics listed in section can be used fo the first two points, while for the third a common choice is to use the p-value of the fit, where the null hypothesis of the test is that the data is originating from the chosen distribution. The aim in this context is thus to look for high p-values, or at least p-values that do not lead to rejecting the null hypotheses (according to the usual threshold of 0.05).

Unfortunately, the p-value suffers from sensitivity to the sample size, and even tiny deviations from the assumed distribution can result in extremely small p-values [52]. Therefore, in real-world cases where the reference dataset to be used for assessing the normality can be composed by thousands of data points, the p-value is perhaps not the best choice.

Within the alternative solutions for evaluating the goodness of fit, a particularly interesting one is the Akaike Information Criterion (AIC), which is based on the maximum likelihood of the model parameters and on a penalty term proportional to the number of the parameters to be estimated. The AIC is thus a relative measure, with no direct probabilistic interpretation, and is defined as:

$$AIC = -2log(L) + 2k \tag{4.17}$$

where k is the number of parameters of the model and L the likelihood. Intuitively speaking, the likelihood indicates the relative plausibility of different parameters given the observed data, and for a set of observations $\mathbf{X} = \{X_1, X_2, \ldots, X_n\}$ and given a parameter vector θ, it is defined as:

$$L(\theta|\mathbf{X}) = \prod_{i=1}^{n} P(X_i|\theta) \tag{4.18}$$

where $P(X_i|\theta)$ is the probability (or probability density) of observing each X_i given the parameters θ. For example, in the case of a normal distribution with mean μ and variance σ^2, the formula for the likelihood becomes:

$$L(\mu, \sigma^2|\mathbf{X}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(X_i - \mu)^2}{2\sigma^2}\right) \tag{4.19}$$

The goodness of the error distribution fit can therefore be assessed, in a relative way across various candidate error distributions using the AIC, or, when the distribution is fixed, just using the likelihood.

In order to asses how fit a model is for the data, given all the of the three quantities introduced at the beginning on this section, these have to be combined somehow. If choosing The APE, MaxAPE and AIC, we are therefore looking for fitness function in the form:

$$fitness = f(APE,\ MaxAPE,\ AIC) \tag{4.20}$$

given that higher fitness values have to be intend as better.

While combining such quantities is a relative simple step for the average and maximum error, the AIC is on an arbitrary scale and can assume virtually any value. For this reason, it has to be scaled somehow, and bought in the same scale of the average and maximum errors, for example dividing it by a reference value suitable for the use-case. A concrete example of a fitness function in case of adopting the APE, MaxAPE and AIC with a reference value AIC_ref is given by:

$$fitness = -(MAPE + MaxAPE + AIC/AIF_{ref}) \tag{4.21}$$

or, if looking for some more meaningful numbers:

$$fitness = \frac{(1 - MAPE) + (1 - MaxAPE) + (1 - AIC/AIF_{ref})}{3} \tag{4.22}$$

4.4.3 Hyperparameter optimization

Hyperparameters are those parameters, specifically in the context of ML and DL, which have to be set before the learning starts. With respect to neural networks, hyperparameters include the learning rate, the batch size, the number of epochs, regularization parameters and perhaps most importantly the type and number of layers and their characteristics, as the number of neurons.

Since such parameters cannot be learned during the training, they have to be set manually. The process of refining their choice and tuning their values in order to get more and more accurate predictions is referred to as *hyperparameter optimization*.

Many techniques exists for hyperparameter optimization, ranging from simple "brute force" approaches to more sophisticated ones, which are briefly described in the following.

Grid Search: hyperparameters are searched enumerating any possible choice of their combinations.

Random Search: instead of testing every combinations, only a subset of them is tested, chosen randomly.

Bayesian Optimization: a probabilistic model of the objective function is built, assuming a continuous search space.

Gradient-based Optimization: if the hyperparameters are differentiable, then the gradient descend algorithm can be used to adjust them based on how the loss changes.

Evolutionary/Genetic Algorithms: a first set of models is created using random values fo the hyperparameters, which is then evolved simulating biological natural selection by assessing the survival rate with a fitness function.

While any of the above methods are suitable for hyperparameter optimization, methodologies that make some informed decisions as the Bayesian or gradient-based optimization and the evolutionary algorithms are to be preferred when possible, since they converge faster.

4.5 Benchmarking

As introduced in section 3.3, anomaly detection techniques cannot be effectively evaluated because of the intrinsic novelties that anomalies represent. This is a key limitation in defining an experimental validation setup. Nevertheless, some benchmarks for anomaly detection have been released, which can provide, to some degree, and indication about how algorithms can perform on *known* anomalies, and in a *given* ground truth.

4.5.1 Available benchmarks and common flaws

As of today, there are just a few publicly available benchmarks for time series anomaly detection, which are summarized in the following. Most of benchmarking toolkits are based on a combination of these well-known benchmarks [64, 86], or make use, possibly alongside such benchmarks, of generative approaches to generate synthetic data on the fly [87].

NASA [42] United State's National Aeronautics and Space Administration released two public datasets for anomaly detection on time series: SMAP (Soil Moisture Active Passive satellite) and MSL (Mars Science Laboratory rover). These are dataset with various quantities as radiation, temperature, power, computational activities, etc.

OMNI [78] This benchmark builds on top NASA's SMAP and MSL benchmarks and introduces the SMD (Server Machine Dataset) benchmark, a 5-week-long dataset in the IT domain collected from a large Internet company.

NAB [1] The Numenta Anomlay Benchmark is a diverse collection of data, primary focused on the IT domain. It includes server data, advertisement clicking rates, and social media trends; but also transportation data and a few air temperature examples. It provides some synthetic data as well.

Yahoo [60] This is a benchmark in the IT domain which contains a mixture of real and synthetic data. It is based on the performance metrics of various Yahoo services.

UCR [88] The University of California Riverside anomaly benchmark is a diverse benchmark mainly based on human medicine and biology data, with a small fraction of the data in meteorology and industry domains. The benchmark slightly changes the angle of anomaly detection benchmarking, framing it as a localization problem rather than as a real detection problem: for any time series in the test dataset, it is known that it contains exactly one anomaly.

Exathon [43] This is a benchmark for high-dimensional time series (i.e. multivariate with thousands of dimensions) in the IT operations domain, which has been systematically constructed based on real data traces from repeated executions of large-scale jobs on an computing cluster.

ESA The European Space Agency anomaly detection benchmark is an extensive benchmark in the satellite telemetry domain. It presents several challenges, as high dimensionality and volume (years of recordings from up to thousands of channels per satellite), a complex network of dependencies between channels, complex characteristics (e.g. varying sampling frequencies, data gaps and losses, trend and concept drifts), and noise and measurement errors due to the influence of the space environment. The benchmark also distinguish between "rare nominal events" and anomalies, with the intention to let algorithms learn from such rare events in the training data, which should not then be flagged as anomalies.

Most of these benchmarks share being heavily flawed, as explained by the authors of the article titled "Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress" [88] mentioned in the introduction of this thesis work. In such article, the authors classify the flaws in four main categories:

1. **Triviality** Trivial anomalies are defined as anomalies that can be found by a single line of code, using only "primitives" as mean, max, std, diff etc. For example, a data point with a value much bigger than the others, or a signal dropping to zero.

2. **Unrealistic anomaly density** This happens when there are either: too many anomalies in the data, contiguous regions marked as single multiple anomalies, or anomalies extremely close each other.

3. **Mislabeled ground truth** Given that anomalies are an ill-defined concept, there can many ground truths, but when a benchmark assumes one, then labeling of the anomalies should be consistent. Mislabeling of anomalies can affect both false positives and false negatives rates, thus invalidating the results.

4. **Run-to-failure bias** This is when the dataset includes the data after a failure. This causes both an unrealistic anomaly density and introduces a bias in terms of anomalies being more likely towards the end of the data. Moreover, including the data after a failure in an anomaly detection benchmark is just an unrealistic setting: once a failure occurs, anomaly detection is not relevant anymore, and metrics computed on such portion of the data are not much informative.

In addition to these flaws, other flaws have been identified in the context of this thesis work, and in particular with respect to sensors-based critical systems. These are numerated below, continuing the list:

5. **Only uni-variate data** While on one hand it is more challenging to detect anomalies in uni-variate data because of the limited informative power they carry, on the other, how to handle the multiple information sources that multi-variate time series provide, and how to combine the multiple "hints" of potential anomalies, are yet other challenges and should be included in a benchmark.

6. **Improper timestamp handling** This is when the timestamps are not properly recorded, or when they are completely missing, thus treating time series data just as generic sequential data. In any time series depending on human activity, earth rotation, or its orbit (thus causing daily or seasonal patterns), the timestamp is fundamental in the modeling process. Moreover, properly handling the time zone is of paramount importance, given that otherwise it might cause a one-hour offsetting at each daylight saving time change.

7. **No data losses** If the benchmarking dataset has no data losses at all, which is hardly the case in all sensor-based systems, then it is not much representative for such systems. Data losses can easily mislead algorithms and lead to several false positives, and should therefore be present in the benchmarks to test how a given methodology or technique would work in a real-world scenario.

8. **Mainly artificial anomalies** This is when most of the anomalies are artificial, i.e. created by a human which modifies the data. Artificial anomalies have limited representative power with respect to real-world anomalies, and inevitably include the bias of the person who created them. Moreover, if the shape of the anomalies is already known, then it would perhaps more interesting to consider a supervised approach, possibly using data augmentation.

9. **Binary anomaly identification** The common practice in benchmarking anomalies is to use a binary approach: data instances are marked either as entirely anomalous, or as entirely not. This approach has several downsides: first of all, it indirectly makes the threshold setting one of the main parameter which gets benchmarked. Secondly, anomalies that get

"almost" detect are dismissed, thus leading to scenarios where a simple threshold adjustment might overturn the benchmarking results. Third, it does not allow to test how good is a given algorithm to assign different magnitude levels to the potential anomalies, a key requirement in several scenarios as introduced in Section 2.4.2. Therefore, it would be highly desirable for a benchmark to includes at least some magnitude classes (small anomaly, medium anomaly, and large anomaly), if not a continuous anomaly ranking.

All of the benchmarks presented above show one ore more of these flaws, the mapping being summarized in table 4.2.

	Critical Flaws (1-4)	Additional Flaws (5-9)	Requires real-time	Includes sensor data
NASA	1, 2, 3, 4	uninvestigated	no	yes
OMNI	1, 2	uninvestigated	yes	yes
NAB	1, 2, 3	uninvestigated	yes	yes
Yahoo	1, 3, 4	uninvestigated	no	no
Exathon	1, 2	uninvestigated	no	no
UCR	none	5, 6, 7, 8, 9	no	yes
ESA	none	9	yes	yes

Table 4.2: Common flaws of time series anomaly detection benchmarks.

Flaws 1-4 identified by the authors in [88] are so critical that their recommendation is to completely abandon the NASA, OMNI, NAB, and Yahoo benchmarks. The authors state: *"As we have demonstrated, they are irretrievably flawed, and almost certainly impossible to fix, now that we are several years past their creation. Moreover, existing papers that evaluate or compare algorithms primarily or exclusively on these datasets should be discounted (or, ideally reevaluated on new challenging datasets)."*

Based on these indications, many works are practically irrelevant, including one of the most comprehensive and recent studies which evaluates several anomaly detection techniques [71].

The UCR benchmark was released by the same authors, with the aim of addressing the flaws they found. The authors place a strong emphasis on the UCR benchmark not being designed to solve them entirely, but rather to mitigate them. However, in the context of this thesis work, also the UCR benchmark appears to have several flaws, as reported in table 4.2.

Moreover, the UCR benchmark frames the anomaly detection problem as a localization problem rather than as a real detection: the benchmark asks, *knowing that there is exactly one anomaly for each time series*, to locate it within a given margin. This benchmark thus does not exactly address real-world scenarios, where the main question is if an anomaly is present or not.

The UCR benchmark also does not require to work in real-time: all of the data can be processed *together* in order to spot where the anomaly is. While this is surely interesting for ex-post analysis as data mining applications and medical exams evaluation, it is of little value in the context of critical systems, where the anomalies must be detected timely, without any information about future data. The UCR benchmark is thus implicitly assuming an unsupervised-like approach to anomaly detection, where, as explained in section 3.2, for each dataset there is at least one anomaly, by definition.

The ESA benchmark is instead nearly flawless, except form the binary anomaly identification, which is however somewhat secondary. The 87-page article presenting the ESA benchmark also indirectly reinforces the importance of flaws 5-8, clearly stating how their data presents all of these challenges.

Unfortunately, while the ESA benchmarks is far the most suitable one if following the benchmarking route, it was released only in the summer of 2024, towards the very end of this thesis work. Given the complexity and the technical challenges it presents, it was not possible to just plug-in the proposed methodology, and indeed the authors openly state that they consider their benchmark as *"a starting point for the development of better algorithms for satellite telemetry anomaly detection"*. In a way, the ESA benchmark opens a new chapter of anomaly detection in critical systems.

4.5.2 Micro-benchmarking results

Given the critical flaws of most of the time series anomaly detection benchmarks as presented in the previous section, and the impossibility of using the ESA benchmark, the compromise made in this thesis was to test the proposed methodology on a subset of the UCR benchmark. However, it has to be kept in mind that such benchmark address a different problem (localization of anomalies rather than real detection), and without any real-time constrains.

One of the main pillars of the proposed methodology is to rely on properly modeling the normal data, thus domain knowledge is essential. In the UCR benchmark, as the authors state, *"the distribution across the domains is highly imbalanced with around 64% of the times series being collected in human medicine applications, 22% in biology, 9% in industry and 5% being air temperature measurements"*. This translated in leaving out most of the data, and in particular all of the data in human medicine (mainly electrocardiogram and respiratory data) and biology (mainly insects EPG[2] and biomechanics force plate and tilt table data), since no experts in these domains where involved.

Moreover, electrocardiogram (ECG) and respiratory data blur the line between signals and time series, which are defined, at least in this thesis work, as a sequence of data over time where *the specific timestamp with respect to a given reference time system of each observation carries important information as well* (Section 2.3). In ECG data, the timestamp does not carry any information if not for the sequential dependency: there is no time reference system

[2]Electrical Penetration Graph

nor the need to know when the ECG was acquired. With the same logic, also an audio file would be a time series.

Also the concept of "real-time" gets blurry in this context: on signals which are sampled in the range of 250 Hz to 1000 Hz, as for ECG and respiratory data, an anomaly can be detected with a delay of hundreds of samples and still be considered as a real-time detection (or at least, as near real time). In case of telemetry data recorded every few minutes or hours, as for the majority of critical systems, this is just not the case. Most of anomaly detection techniques for ECG data do not rely on forecasting indeed, but rather on reconstruction, using hundreds of samples simultaneously, if not the entire signal.

Along the same lines, the UCR benchmark allows for a 100-sample tolerance in detecting the anomalies. While for ECG and respiratory data this translates to a fraction of a second, for data sampled at one-hour intervals it translates instead to 100 hours. This shows yet another flaw of the UCR benchmark: on the data sampled at one-hour intervals which is part of the benchmark, as it will be shown in moment, an anomaly detected more than four days later (or earlier) it is still classified as correctly detected.

After discarding data in the medicine and biology domains, only two types of data were then left: in the industry domain, which is mainly power demand, and the air temperature data. Again due to the strong bias of this benchmark towards ECG and respiratory-like data, there are no timestamps at all, nor information about the sampling interval. In order to inspect the data for power demand and air temperature, the sampling interval was quickly reverse-engineered and discovered to be in nearly all cases of one hour. Figures 4.9 and 4.8 show an example fo the "normal" data for the power demand data and the air temperature data.

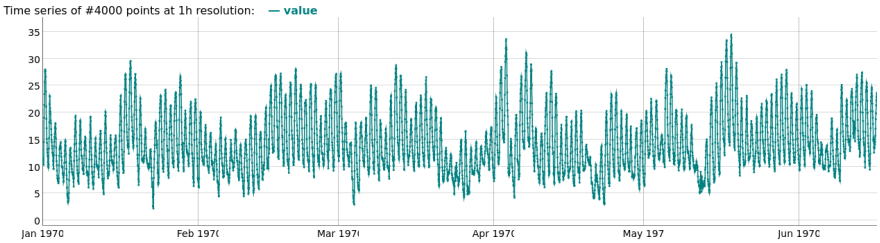


Figure 4.8: Air temperature "normal" data of the UCR benchmark.

If visually inspecting the power demand data, a spike clearly stands out, towards the end of the time series. This is zoomed in Figure 4.10, where it can be seen that it occurred exactly on the year change. It is hard to believe that this not due to a data acquisition or processing issue, but regardless of the root cause, this should definitely count as an anomaly, while the UCR dataset states that the "normal" data can be assumed free from anomalies.

Moreover, power demand has a strong dependency on "human" time, and modeling this kind of data without any information about the hour of the day, the day of the week, or if a day is a bank holiday or not, it is just unrealistic.

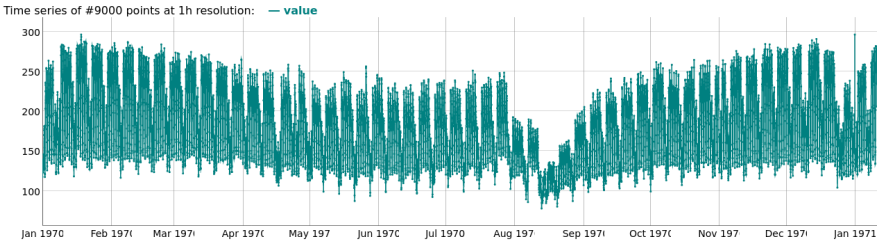


Figure 4.9: Power demand "normal" data of the UCR benchmark.

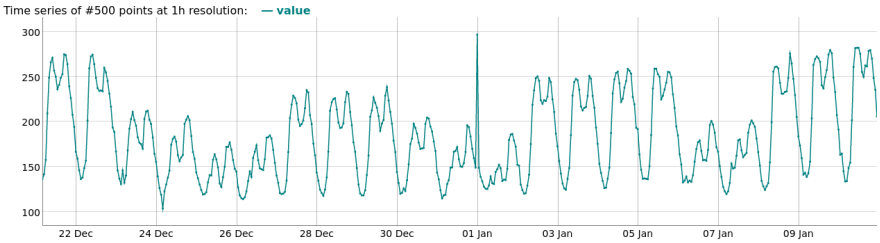


Figure 4.10: Power demand "normal" data of the UCR benchmark, zoomed on the first part.

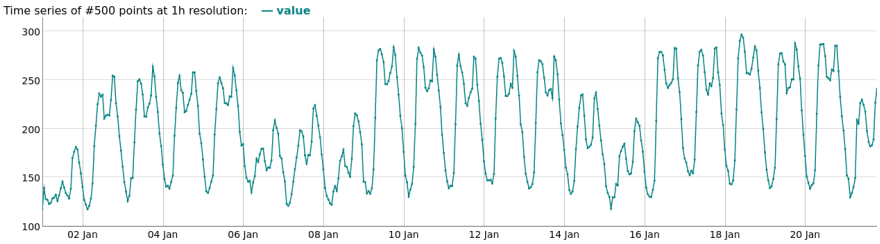


Figure 4.11: Power demand "normal" data of the UCR benchmark, zoomed on the first part.

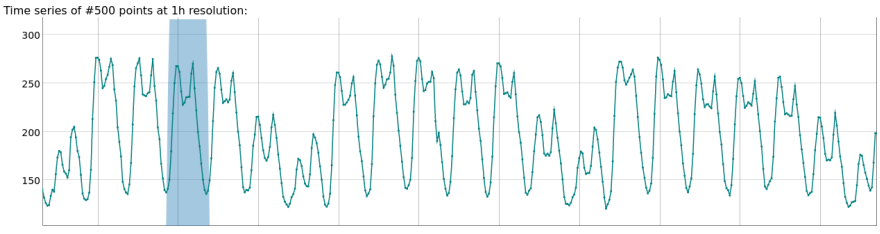


Figure 4.12: Power demand "test" data of the UCR benchmark, zoomed on one of the anomalies marked by the authors (highlighted in blue).

At this point, the power demand data was considered to be flawed enough to be discarded, and the anomalies marked by the authors inspected, out of curiosity. Interestingly enough, one of them just removes a day from a week, as it can be clearly seen in Figure 4.12, where the anomaly marked by the authors is highlighted in blue. There are several conceptual issues with this "anomaly". First of all, the anomaly is the day being removed, therefore it would technically not even be possible to represent it. If the authors marked the day after the day being removed, then why not marking the day before, or both? Or, why not marking the entire week? In second place, this kind of pattern (a week with four days) already presented itself at the beginning of the "normal" data because of Epiphany, so it is unclear how a model (or a even a human) should be able to capture the difference: the day removed could have easily been four days after, on a Sunday, and both statements would be correct. Lastly, all the data after the anomaly is shifted by one day, therefore also an algorithm marking the entire time series as anomalous after the missing day could be right. On top of these issues, such anomaly is of little interest for real-world scenarios, where potential data acquisition issues are handled with simple conditional checks and where timestamps are handled correctly, so that a day does not just "disappear". In summary, this is yet another example about all the potential pitfalls that can arise in benchmarking anomaly detection techniques, and why domain knowledge is essential.

The only data left is then the air temperature data, which comprises 13 time series (and 13 anomalies), an thus constitutes the micro-benchmark on which the proposed methodology has been tested.

Three forecasting models have been trained, all based on a LSTM neural network, to compute and then applied computing the discrepancies between the predicted and actual values, which were then in turn used to compute the anomaly index as per Section 4.2.1. The first model was setup to predict the next data point using a previous window of data, the while the second and the third were setup to have some samples of the last part of the window removed (respectively, two and four), in order to prevent the model to adapt too much on new patterns.

Only the data values, without the timestamps, were used for the models, as per benchmark specifications. The error metric of choice for evaluating the discrepancies between the predicted and actual values was chosen as the Percentage Error (PE), because air temperature errors should be scale-invariant.

Hyperparameters (just the number of neurons and the window length in this case) have been optimized with a simple grid search, choosing the best trade-off between the Mean Absolute Percentage Error (MAPE), the Maximum Absolute Percentage Error (MaxAPE), and the AIC (as introduced in Section 4.4).

Since the benchmark does not require to asses the presence of anomalies but instead to identify the location of the bigger anomaly, the problem was framed similarly to an unsupervised anomaly detection problem. The anomaly index lower bound was then set to 3 standard deviations, and the upper bound to an adherence value of $1/10^9$, following the reasoning presented in Section 4.2.2.

The anomaly indexes originating by the different models were then merged using the maximum value, and grouped into events with a tolerance of a single point gap (that is, if two anomaly events were separated by a gap of a single data point, they were merged together), where the event magnitude was defined by the average value of the anomaly index across the entire event.

Anomalies were then counted as correctly identified only if the bigger anomaly event was overlapping the anomaly marked by the authors, thus without the 100-sample tolerance, which as previously explained would not make much sense on such data.

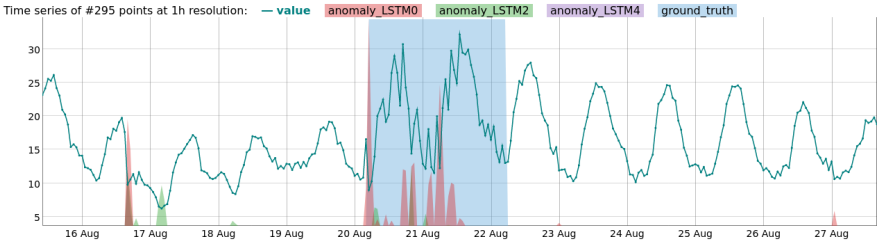


Figure 4.13: Example results on the air temperature data of the UCR benchmark with anomaly index breakdown on a per-model basis. LSTM0 is the model using the entire widow, while LSTM2 and LSTM4 are the models which had, respectively, two and four samples removed from the last part of the window.

Of the 13 anomalies, 8 were correctly identified, leading to an accuracy of 0.61. An example of an anomaly correctly detected is shown in Figure 4.13, together with the anomaly index breakdown on a per-model basis. Recent state of the art benchmarking results on the entire UCR dataset [51] report accuracy of 0.71 (TimeVQVAE-AD), with a second place accuracy of 0.51 (Matrix Profile STUMPY) and third of 0.47 (MDI). The top ten performing techniques on the entire UCR dataset are reported in Table 4.3.

Given that the excluded data could have been potentially much more challenging, the TimeVQVAE-AD accuracy was re-evaluated only on the air temperature data, using the original published results. This showed an increase in the accuracy, that raised to 0.84, whch is directly comparable to the accuracy of 0.64 obtained by the proposed methodology (Table 4.4).

An accuracy below state of the art was expected, given the different nature of the benchmark with respect to the use-case that the proposed methodology aims to address. It has indeed to be remarked that the techniques reported in Tables 4.3 and 4.4 do not work in real-time, but have visibility of the entire time series, including the data after the anomaly. The comparison is thus inherently unfair with respect to the proposed methodology.

On the other hand, and bearing in mind the limited informative power of this micro-benchmark, it still seems to fall in the same ballpark as the best performing ones.

Technique	Accuracy	Data
TimeVQVAE-AD	0.71	entire dataset
Matrix Profile STUMPY	0.51	entire dataset
MDI	0.47	entire dataset
Matrix Profile SCRIMP	0.41	entire dataset
RCF	0.39	entire dataset
IF	0.38	entire dataset
Convolutional AE	0.35	entire dataset
SR-CNN	0.30	entire dataset
USAD	0.28	entire dataset
AE	0.28	entire dataset

Table 4.3: Benchmarking results on the UCR anomaly detection dataset.

Technique	Accuracy	Data
TimeVQVAE-AD	0.84	air temperature
Proposed methodology	0.61	air temperature

Table 4.4: Mmicro-benchmarking results on the UCR anomaly detection dataset.

Chapter 5

The Timeseria software library

Timeseria is an object-oriented time series processing library implemented in Python, which aims at making it easier to manipulate time series data and to build statistical and machine learning models on top of it.

Unlike common data analysis frameworks, it builds up from well defined and reusable logical units (objects), which can be easily combined together in order to ensure a high level of consistency.

Thanks to this approach, Timeseria can address by design several non-trivial issues often underestimated, such as handling data losses, non-uniform sampling rates, differences between aggregated data and punctual observations, time zones, daylight saving times, and more.

Timeseria comes with a comprehensive set of base data structures, common data manipulation operations, and extensible models for data reconstruction, forecasting and anomaly detection. It also integrates a powerful plotting engine capable of handling even millions of data points.

5.1 Motivation and significance

Time series are central to several fields across many disciplines, such as engineering, meteorology, medicine, environmental science, finance, economics, and more. Time series represent the evolution of a phenomena over time, and their analysis is essential to capture the dynamics of the phenomena being studied, understand cause-and-effect relationships, and make predictions.

However, a typical time series processing pipeline — loading a dataset, cleaning and plotting it, performing some operations, applying some models and inspecting the results — still feels unnecessarily cumbersome. Scientists, engineers, analysts, and many other professionals spend a considerable amount of time on repetitive tasks and on getting their code to work, instead of focusing on their core analysis.

Current numerical and data analysis frameworks indeed fall short in handling everyday practical challenges when it come to time series data, and perhaps most importantly, in real-world scenarios.

In a recent review [75] that surveyed 40 time series processing packages, within the 17 listed as capable of handling missing data, upon closer investigation it can be seen that none of them provides a robust solution, but more of a set of workarounds. Similarly, among the packages listed as capable of performing modeling tasks, checking for data consistency is always left to the user.

It is also important to note the general lack of definitions regarding what constitutes time series data, and how to represent it [16]. This is partly because each format has it own advantages and disadvantages, and partly because the complexity of such data is often underestimated.

Simple vector or matrix data structures are usually the default choice, until discovering at later stages that are just not capable of taking into account most of the typical time series characteristics: data losses are represented ambiguously, calendar arithmetic and daylight saving times are handled poorly, there is no support for variable sampling rates and no clear distinction between punctual observations and aggregated data.

In this context, the usual solution is a series of patches stuck together which eventually backfires, and very few attempts have been made to get the fundamentals right. The TSflex [84] package, which focuses on time-series preeprocessing and feature extraction, is one of such rare examples.

Timeseria represent an effort towards this goal: it is designed using a novel, object-oriented approach for representing time series data, which can address most of the issues outlined above.

5.2 Related work

Within the large body of work on libraries, tools, and frameworks for data processing suitable for time series data, we selected the most relevant ones with respect to Timeseria. The key criterion was their usability as building blocks (i.e., as a library), which excluded, among others, GUI (Graphical User Interface) applications, web-based solutions, and closed-source software.

One of such examples is Wolfram Mathematica, a commercial (and closed-source) software solution originally developed for symbolic computation, which provides several functionalities for time series data. Although these functionalities could, in principle, be compared to Timeseria's, the comparison would be challenging on both sides, given their different nature, goals, and target audiences, as discussed in a GitHub issue opened on Timeseria's repository[1]

Also the R [21] software, which is instead open source, provides functionalities for time series processing that could be compared to Timeseria's. In particular, R provides the `ts` function, which allows to create and manipulate time series objects. In this case as well, a direct comparison would be difficult,

[1] `https://github.com/sarusso/Timeseria/issues/40`

in particular because R is designed for interactive analysis and data visualization, lacking the general-purpose features and ecosystem support provided by other languages, which severely limits its usability as a building block for other software.

In addition, several domain-specific tools are available, often provided in the form of GUI or Web-based applications. Some examples include MaD GUI [62] (a tool for creating GUI interfaces for domain-expert time series annotation), GRETA [57] (a web-based solution that produces hourly wind and solar photovoltaic generation time series), and OXI [69] (an online tool for visualization and annotation of satellite time series data). While such tools can undoubtedly outperform more general-purpose solutions (like Timeseria) for the use case or domain they were originally developed for, they are deeply intertwined with their original purpose, making them difficult to adapt, even if heavily modifying the code.

With respect to the solutions that can instead be directly compared to Timeseria, and in particular in the realm of the data structures, Pandas [56] `Series` and `DataFrames` are the most commonly used for representing time series data in the Python ecosystem. They provide basic support for time series data trough the `DatetimeIndex`, which enables operations and slicing based on time. Resampling functionalities are also supported, with basic handling of data losses, provided that they have been already marked as null values. Xarray [38] is another widely used option in the Python ecosystem, which provides data structures for n-dimensional labeled arrays. Time series data can be represented using the `DataArray` class, which features interfaces similar to Pandas `Series` or `DataFrames` but is optimized for multi-dimensional data and large datasets.

Timeseria provides instead its own data structures for time series data, which are built mainly from scratch. This design choice was necessary in order to achieve the consistency and abstraction levels Timeseria aimed for. Timeseria data structures are indeed not optimized for performance, but for clarity of abstractions and consistency. This approach made it possible to differentiate between punctual observations and aggregated data, to integrate handling of data losses into the library's foundations, to manage time nuances as time zones, daylight saving time, and calendar time effectively, and more.

Higher-level solutions are built almost exclusively on top of Pandas or Xarray data structures, and thus lack the consistency and abstraction levels that Timeseria can provide. In the following we report the most notable ones, which typically focus on modeling aspects.

SKtime [54] and Tslearn [80] are general-purpose Python libraries for a variety of time series machine learning tasks, offering a combination of tools for pre-processing, feature extraction, clustering, classification, regression and forecasting.

TSflex [84] is a Python toolkit for flexible time series processing and feature extraction, claimed to be particularly efficient and making few assumptions about the underlying data. As mentioned earlier, it is one of the few examples

that aim to address the fundamentals properly, albeit with a narrow focus.

GluonTS [2] is a Python package for probabilistic time series modeling focusing on deep learning models, and AutoTS [47] is a package designed for rapidly deploying high-accuracy forecasts at scale, testing several forecasting models using a genetic approach, and finding the best candidates autonomously.

Prophet [82] is instead defined as a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, along with holiday effects, which is implemented in both R and Python.

Greykite [36] aims to provides flexible, intuitive and fast forecasts through its flagship algorithm, Silverkite, which is particularly indicated for time series with change points in trend or seasonality, event/holiday effects, and temporal dependencies.

PyOD [94], is a comprehensive Python library for detecting outliers and/or anomalies in multivariate data, while Orion [3] is a machine learning library built for unsupervised time series anomaly detection, mainly using Generative Adversarial Networks, with the goal of identifying rare patterns and flag them for expert review.

With respect to time series modeling, Timeseria focuses less on providing a wide selection of models, and more on providing well-defined building blocks that allow users to plug-in their own modeling logic with minimal effort, possibly integrating with all of the above solutions (Prophet is indeed used as an example built-in model). The goal of this choice is to bring together the best features of the aforementioned packages with the unique strengths of Timeseria.

Another class of related work are libraries and frameworks targeting the entire data analyses pipeline, which are generally (if not entirely) mutually exclusive with Timeseria. We identified three of them as the most relevant ones: Darts, Kats, and ETNA.

Darts [34] is a Python machine learning library for time series, with a focus on forecasting. It offers a variety of models, from classics such as ARIMA to state-of-the-art deep neural networks. It provides a dedicated main class `TimeSeries` which stores data as a `DataArray` (from the Xarray package).

Kats [26] is a light-weight, easy-to-use, extendable, and generalizable framework to perform time series analysis in Python. It aims to provide a one-stop shop for techniques for univariate and multivariate time series including forecasting, anomaly and change point detection, feature extraction, and more. Kats provides a dedicated `TimeSeriesData` structure to represented univariate and multivariate time series, which is based on a tuple of Pandas `Series`, one for the timestamps and one for the values.

ETNA [14] is a time series forecasting framework which aims at ease of use. It includes functionalities for time series pre-processing, feature generation, several forecasting models with a unified interface, model combinations and evaluation trough back-testing. It makes use of a `TSDataset` format for representing time series data, which is a wrapper around a Pandas `DataFrame`, offering several functionalities for time series data.

While all of the work presented in this section could be compared to Timeseria to some extent, only Darts, Kats and ETNA show significant overlap and can thus be compared in a direct way. We therefore conducted a comparative analysis to highlight their respective differences, with particular attention to Timeseria unique functionalities but also from other angles, that is presented in Table 5.1.

	Timeseria	**Darts**	**Kats**	**ETNA**
Dedicated, object-oriented base data structures	yes	no	no	no
Differentiation between aggregated and punctual observations	yes	no	no	no
Extensive time zone and calendar time support	yes	no	no	no
Extensive support for data losses	yes, with auto-detect	no	no	no
Dedicated, interactive plotting	yes	no	no	no
Forecasting	yes	yes	yes	yes
Reconstruction	yes	no	no	no
Anomaly detection	only model-based	with several techniques	only as outliers	only as outliers
Model apply [1]	yes, for all models	no	no	no
Evaluation and cross-validation[1]	yes	only as utility functions for the residuals	no	yes, as pipeline back testing functionality
Support for custom models	yes, with several functionalities	limited, no significant functionalities	limited, no significant functionalities	yes, with several functionalities
Built-in models	few, mainly as examples	several	several	several
Probabilistic support	under development	yes	only for forecasts	only for forecasts
Computing performance[1]	lower than standard	standard	standard	standard

Table 5.1: Comparison of Timeseria with similar solutions (Darts, Kats, ETNA). [1] Such methods are intended as functions that perform their task in just one call, without requiring any extra code. [2] By "standard" performance it has to be intended the performance that can be obtained when relying on Pandas or Xarray data structures.

5.3 Architecture and functionalities

Timeseria presents itself as a Python library that can be imported and used in interactive data analysis environments (as the Jupyter Notebooks), batch data analysis pipelines as well as in more structured projects. Its features work in both environments, e.g. the logger is fully customizable through handlers and the plots can be rendered either as interactive applets or static images.

As previously mentioned, Timeseria provides set of objects that can be easily combined together, the most basic one representing a simple point in a space. This is then declined in a point in time, and in a point with some data attached. Combining the two leads to a data point in time with some data attached, which is one of the data structures most widely used in Timeseria.

In order to represent aggregated data, Timeseria introduces the *slot* concept. A slot ranges from a point to another and naturally defines a *unit*, which corresponds in first approximation to its length. Both points and slots can have data attached. Similarly as for the points, also the slots can live in time and have some data attached. However, slots do not carry pure observational data, but always aggregates: an average temperature, a count of cars, an energy, an average stock price.

Series are defined as sequences of points or slots, with the added requirements for slot series to be "dense": slots must be in succession, with no gaps. This is one of the main structural requirements of Timeseria, and is based on the concept that while points represent samples in a space (the time, in this context) which can be acquired anywhere, aggregated data represents instead a discretization: it forms a grid, and therefore cannot have holes.

To manipulate series, Timeseria provides a set of *operations* plus two *transformations*: resampling and aggregation. Operations can return other series or scalar values, and include merging, filtering, slicing, rescaling, offsetting, integrating and differentiating as well as computing summary values as the average, the minimum and the maximum, and more.

The resampling transformation creates new, evenly spaced data points making use of a given interpolation strategy in order to fill gaps due to missing samples. The aggregation transformation applies instead one or more aggregation operations, and if applied on sparse point series it first resamples it. Any operation returning a scalar value can be used in the aggregation (the average being the default one). Both transformations also compute the data loss for each point or slot.

Timeseria also provides a base class for generic models, and specific base classes for *forecasters*, *reconstructors* and *anomaly detectors*. These provide common functionalities as fitting, applying, evaluating and cross validating a model. Some specific implementations are provided, as simple periodic-averages, auto-regressive moving averages (ARIMA), automatic procedures as Prophet and Deep Learning-based ones as LSTM neural networks. However, Timeseria provides such implementations more as examples, and while they can already provide interesting forecasting, reconstruction and anomaly detection capabilities for basic tasks, the intent is to let users easily extend the base

model classes plugging n their own implementation (while relying on Timeseria for the common functionalities).

The following diagram represent the main classes that reflect the concepts outlined above, and their compositions:

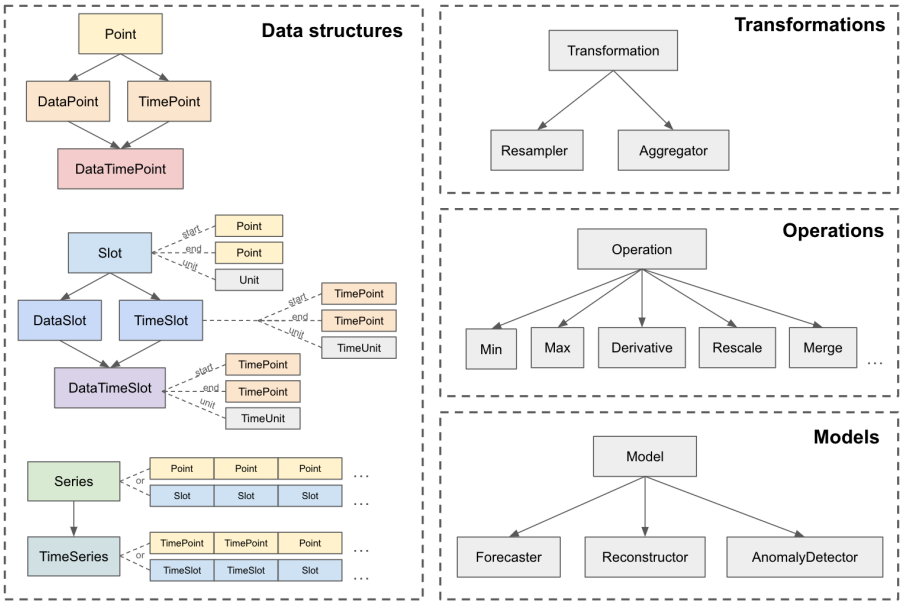


Figure 5.1: Timeseria base classes structure

Other modules include the Units module, where slot units are defined; the Plots module, for the plotting engines; the Storage module, for the data storages; the Interpolators module, where interpolation strategies are defined; the Utilities module, which provides common functions for utility tasks; and the Exceptions module for custom errors.

In the following, classes are represented in mono-space font using camel case (e.g. `MyClass`) while methods, functions, attributes and variables are represented in mono-space font using lower case letters and underscores (e.g. `my_function()`, `my_variable`).

5.3.1 Datastructures

The `datastructures` module provide all the base data structures: `Points`, `Slots` and `Series`, together with their specializations.

A `Point` is just a point in any n-dimensional space, as for example given by the coordinates 1,2. It supports common mathematical operations with respect to other points or units. A `TimePoint` is a point in the time dimension, measured in epoch seconds, where the zero corresponds to the midnight of 1st January 1970, UTC. Sub-second precision can be achieved with decimals. Time points support being initialised both using epoch seconds (with the `t` argument) and `datetime` objects (with the the `dt` argument). They also provide the

respective attributes for ease of use. Time points support arithmetic operations with respect to other time points, time units, epoch seconds (as floating point variables) and with `datetime` objects. This makes it very easy to navigate along the time axis, so that for example a time point can be easily shifted one month in advance by just adding a one-month time unit. A `DataPoint` is an extension of the point concept, with some data attached. This can be any kind of valid Python data, however Timeseria works best with list-based or dict-based data. To then define data points is time, Timeseria merges `DataPoints` and `TimePoints`, creating the `DataTimePoints`.

`DataPoints` and `DataTimePoints` can also have a "validity region" attached, which defines where their data should be considered as representative. If not set, this is usually automatically computed according to the context. If set, this allows to account for variable sampling rates, and most importantly to discriminate between an increased sampling interval and a data loss.

While points represent punctual observations, `Slots` are instead defined from a *start* to an *end*. Slots are naturally associated with a *unit*, which represents the slot span. Slot start and end are `Point` objects, while the unit is an `Unit` object. Similarly to the points, also for the slots Timeseria defines the `DataSlots`, `TimeSlots` and most importantly the `DataTimeSlots`. The triplet start-end-unit must be consistent, and only two of them are required to be set. For example, to create a one-day slot containing the average temperature value for the 21st November 2022, this can be achieved creating `DataTimeSlot` with a start set to a `TimePoint` for the 21st November 2022 and a slot unit set to a `TimeUnit` object of "1D" (one day). The slot end is then automatically computed (and vice-versa).

Both points and slots support a `data_loss` attribute, which is a key feature in Timeseria. This represents the data loss occurred when computing a new point in resampling (i.e. because its nearest neighbor was too far), or when computing a new slot in aggregating (i.e. because there were missing punctual observations for that slot). Keeping track of data losses is important when deriving conclusions from data or when fitting a model, to prevent misleading conclusions in the first case and the so called "garbage in - garbage out" issue in the second.

The data loss is a particular case of a more generic concept: the *data indexes*. These are indicators in the 0.0 - 1.0 range of some property of the data, attached to each data point or slot. Example data indexes include, besides the data loss, the *data reconstructed* index, the *forecast* index, and the *anomaly* index. Data indexes are recomputed when resampling or aggregating, so that they are correctly carried on.

`Series` are defined as a list of items coming one after another, where every item is guaranteed to be of the same type and in an order or succession. Ordering is checked with the standard comparison operators, while it is up to the series items to define a succession logic, if any. In Timeseria, points never define a succession, while slots always do. In other words, it is enforced that if data is slotted, it then must be dense, without any hole.

Series can virtually contain any data type, but Timeseria put its focus on `(Data)Points` and `(Data)Slots`, and in particular on `(Data)TimePoints` and `(Data)TimeSlots`, which define the `TimeSeries`. Series and time series support a variety of manipulation operations, as slicing, filtering, duplicating, extending, and several mathematical operations, as it will be explained in Section 5.3.2. They also support two transformations: resampling and aggregating, which will be introduced in subsection 5.3.3.

Time series also have a *resolution* attribute, which represents the temporal resolution. This concept is similar to a sampling interval, but generalised in order to make it compatible with slots as well. For example, a point time series sampled at 1-minute intervals has resolution corresponding to the time unit "60s", while a one-day slots time series has resolution corresponding to the time unit "1D". Time series with a variable sampling rate (e.g. due to a non-constant sampling process) report instead "variable" as their resolution.

Lastly, one of the main architectural features of Timeseria is to allow time units to have *variable* length. This allows to effectively model months with a variable number of days, and days with a variable number of hours (due to the daylight saving time). It does indeed *not* hold true that a time unit of 24 hours ("24h") is equal to a time unit of one day ("1D") : the first is always exactly 24 hours, while the second can assume different lengths: 24, 24 or 25 hours, depending whether there is an ongoing change in the daylight saving time, and in which direction.

Series and time series can be created in a number of ways, and in particular from a list of items, a Pandas `DataFrame`, or a path to a CSV file (which will in turn use the Storage module to read and parse it). They also provide, among the others, a `to_csv()()` method to save them as a CSV file, and a `to_df()` method to obtain their contents as a Pandas `DataFrame`.

5.3.2 Operations

`Operations` can return another series, a scalar, a list of items, or any other valid data type. An operation, if returning a series, an operation never changes its shape (intended as the resolution), which is a task reserved for the transformations. Timeseria provides several built-in operations, as: `Min`, `Max`, `Avg` and `Sum`, which operate on a series and return scalars; `Derivative` and `Integral` and their discrete counterparts `Diff` and `CSum` (cumulative sum), `MAvg` (moving average), `Normalize`, `Offset`, `Rescale`, `Filter` and `Slice`, which all operate on a series and return another series; `Merge`, which operates on two or more series and return the merged series; and lastly the `Select` operation, which allows to select (and return) only specific items of a series. All operations can also be accessed as series methods, e.g. `series.diff()`.

5.3.3 Transformations

The `Transformation` class represents a generic transformation concept, which if applied on a series changes its *shape*. Timeseria provides two trans-

formations, the `Resampler` and the `Aggregator`. The first transforms a point series into another point series with a different resolution (sampling interval), the second transforms a point series in a slot series, aggregating it accordingly to the given unit. Both of them recreate missing data using an `Interpolator`, which is by default a linear interpolator, and set the `data_loss` attribute of the new points or slots It has to be noted than while the resampling transformation is to be considered an alternative "view" of the original signal, which should be driven by the Nyquist–Shannon sampling theorem, an aggregation computes instead indicators of the underlying signal, the default one being the average. Aggregators can support one or more of these indicators, which are just the operations of the previous module (assuming they return a scalar). Custom operations extending them can also be plugged in. The aggregation operation(s) can be changed simply by setting the `operations` argument when initializing the aggregator, either passing the operations themselves or just their string representations. For example, the aggregator initialized with the statement: `Aggregator('1D', operations=['min','max','avg'])` will aggregate the series in daily slots where each of them will contain the average value together with its minimum and maximum. As for the operations, also the transformations can be accessed as methods of the series object, for example as `series.resample('1h')`.

5.3.4 Models

The base `Model` class can represent both stateless models, where all the information is coded and there are no parameters, and stateful (parametric) models, where there are a number of parameters which can be both set manually or learned (fitted) from the data. All models expose a `predict()`, an `apply()` and an `evaluate()` method. Parametric models also provide a `save()` method to store the parameters of the model, and an optional `fit()` method if the parameters have to be learned form the data. In this case also a `cross_validate()` method is available. Models operating on series also enforce time resolution granularity and data consistency between methods and save/load operations, and require the series to have fixed, non-variable time resolution in order to operate. If this is not the case, for example because the data is not equally spaced over time, or because it has some missing data, then it can be just resampled (or aggregated).

Models are sub-dived in three main categories: *forecasters*, *reconstructors*, and *anomaly detectors*.

- `Forecasters`: this class of models is dedicated to forecasting new data. Forecasters predict n steps ahead, either using their internal logic to predict all of them in one go, or by recursively re-applying the same logic n times, using previously predicted data as if it was actual data.

  Forecasters can operate with or without a window, and the window size can be any value greater or equal than one. Edge cases (as applying the forecaster on a time series with not enough elements) are automatically

handled by the base `Forecaster` class, which also adds the *forecast* data index on foretasted data points (or slots).

Forecasters can be evaluated by making use of one or more error metrics, including custom ones, plotting the error distribution, and visually inspecting the resulting time series. Cross validation is available as well.

Some examples of the forecaster available in Timeseria include the `Periodic AverageForecaster` (a forecaster based on computing periodic averages over historical data), the `LSTMForecaster` (which makes use of an LSTM neural network), and the `AARIMAReconstructor` (which makes use of an auto-ARIMA model).

- `Reconstructors`: this class of models work on reconstructing missing data, or in other words to fill *gaps*. Gaps need a "next" element to be defined, which can bring much more information to the model with respect to a forecasting task. Gaps are identified using the `data_loss` attribute, on both data points and slots. By default, only gaps with a full data loss (e.g. a sequence of slots where there were no underlying data points at all) are reconstructed.

  Reconstructors can operate with or without a window, which can be present before the gap, after the gap, or both. The window size can be any value greater or equal than one. Edge cases (as applying the reconstructor on a series with not enough elements before or after a gap) are automatically handled by the base `Reconstructor` class, which also adds the `data_reconstructed` data index on data points (or slots) that were reconstructed according to the reconstructor policy.

  Evaluating a reconstructors takes in input one or more gap length, and can make use of one or more error metrics. It is not a trivial task as the gap lengths to be used for evaluating a reconstructor should depend on their distribution in the dataset, making it a dataset-dependant task.

  Examples of reconstructors include the `PeriodicAvergeReconstructor` (based on periodic averages of historical data), the `ProphetReconstructor` (which makes use of Facebook's Prophet, an automated modeling procedure), and the `LinearInterpolationReconstructor`, which is particularly useful for benchmarking purposes. The reason why there is a dedicated reconstructor based on an interpolator, instead of just listing the interpolator, is because Timeseria makes a distinction between interpolators and reconstructors: interpolators are used in the transformations, before resampling or aggregating data and must therefore support variable-resolution series, while reconstructors require fixed-resolution series.

- `AnomalyDetectors`: this class of models aim at spotting anomalies sin a given time series. As explained in chapter 3, there are a number of ways to achieve this goal, and as of today, Timeseria implements only model-based anomaly detection.

Besides the generic base `AnomalyDetector` class, Timeseria provides the `ModelBasedAnomalyDetector` class for such task, which implements all of the model-based anomaly detection logic. Any model sub-classing the base `Model` class can be plugged-in, and in particular all of the `Forecasters` and `Reconstructors`, including user-defined ones. Each data point or slot is then assigned with an `anomaly` data index, representing the likelihood of that point or slot of being anomalous, as described in Section 4.2.1.

The difference between using a reconstruct or a forecaster lies in what data visibility the they require: forecaster only require the last data point and can thus be used in a "live" setting, while reconstructors cannot as they always require one (or more) "next" data point(s). Besides these two base class, Timeseria provides some pre-assembled anomaly detectors, and in particular the `PeriodicAverageAnomalyDetector` and the `LSTMAnomalyDetector`, both based on their respective forecasters.

While the above description listed some common model implementations, the true goal of Timeseria is to provide a framework where to easily plug-in custom modeling logic by extending the base classes. Indeed, when extending them, a set of common methods and decorators is automatically inherited, so that only the model-specific logic needs to be coded. For example, the base `Forecaster` class provides the `apply()`, `evaluate()` and `cross_validate()` methods, together with two decorators: `@Forecaster.fit_method` and `@Forecaster.predict_method`, to be used to decorate the custom `fit()` and `predict()` methods. All common functionalities, including the `save()` and `load()` methods as well as the data consistency checks, are thus automatically handled.

5.4 Implementation

Timeseria is implemented in Python 3 and versioned with Git. The source code is available on GitHub[2] (Figure 5.2), and released as Python package on PyPI[3] (the Python Package Index).

All the base data structures are implemented as custom objects, except from the series and the time series which instead extend the Python list object, thus reusing several built-in functionalities. Timeseria is easily extensible by sub-classing the base classes provided in the various modules.

The Plotting engine makes use of the Dygraphs Javascript library, and can generate interactive plots within Jupyter Notebooks and Jupyter Lab, saving them as HTML files (also useful for embedding them in websites or web applications), or as static images. Such engine is capable of plotting even millions of data points thanks to pre-aggregating data: in this case the average value of a set of data points (or slots) is plotted as a line chart together with the minimum

[2] `https://github.com/sarusso/Timeseria`
[3] `https://pypi.org/project/timeseria/`

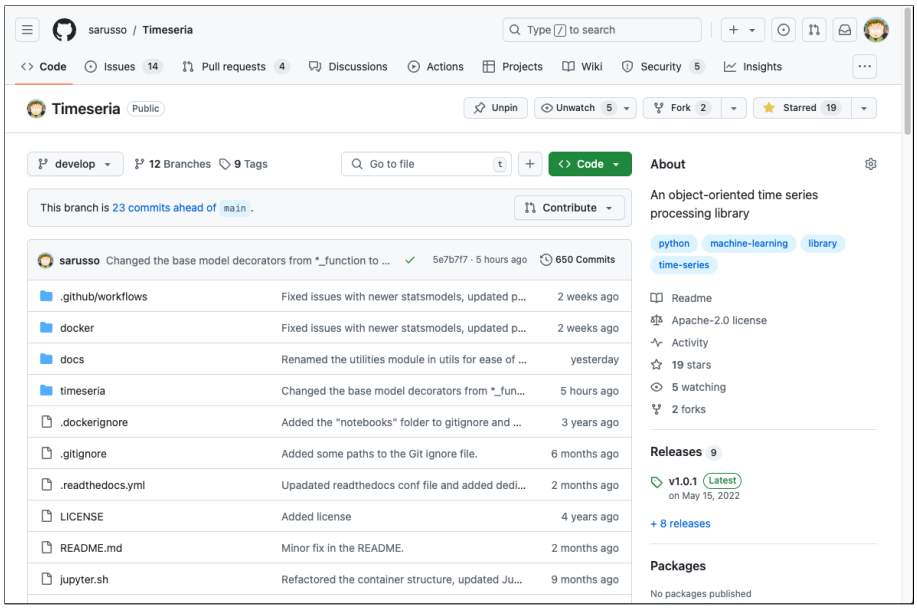


Figure 5.2: Timeseria Git repository on GitHub.

and maximum values as an underlying area chart, so that peaks are still visible even if data is aggregated. A similar approach is implemented in [83], and it is important when visually inspecting large time series. The time is manages using the `propertime`[4] library, which was developed together with Timeseria. Such library makes use of Python built-in `datetime` objects while extending them in order to provide strict time management, as for example not allowing any naive timestamp (without a timezone / UTC offset) and enforcing correct DST changes. It also includes a `dt_range` object which overcomes several pitfalls of Python `timedelta`. Other libraries include the chardet library, used for automatically detecting the encoding of the CSV files, and the Numpy's Fast Fourier Transform (FFT), used for detecting periodicity.

In general, Timeseria is not performance-oriented, but rather consistency-oriented. Its implementation optimizes indeed for consistency over performance, following the assumption that, with today's computing power, for a variety of tasks performance is somewhat secondary, while data consistency is the most important requirement.

Timeseria is relatively well tested, being its development mainly test-driven, and as of today a total of 128 test cases with 1411 assertions ensure a good coverage. Each commit on the GitHub repository is automatically tested using GitHub Actions, in a continuous-integration fashion. Versioning of releases follows semantic versioning (v2.0.0), and the reference Timeseria version for this thesis work is version 2. The documentation is automatically generated using Sphinx and Napoleon, and made available online[5] (Figure 5.3).

[4] `https://github.com/sarusso/Propertime`
[5] `https://timeseria.readthedocs.io/`

Timeseria also comes with a set of example Jupyter Notebooks, available on a GitHub repository[6], and with a Docker container available on Docker Hub[7], which allows for full reproducibility of all the examples.

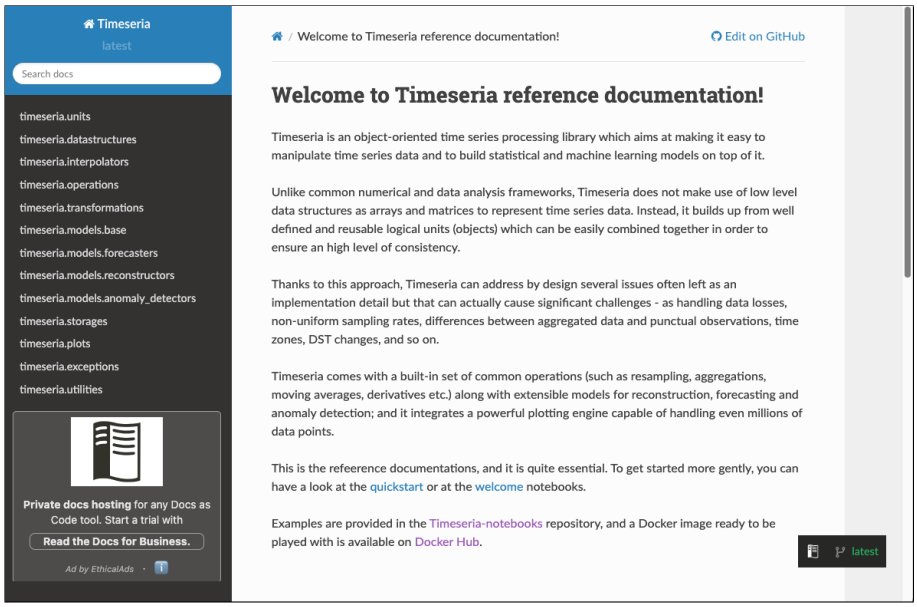


Figure 5.3: Timeseria Sphinx-based documentation on Read The Docs.

5.5 Illustrative examples

The following examples can be executed with a vanilla installation of Timeseria, either in a Jupyter Notebook environment or within a standard Python script. In this case, the plots must be generated as images and saved to a file, adding the following extra arguments to each `plot()` call: `image=True, save_to='plot.png'`. For the `LSTMForecaster`, the extra `tensorflow` optional dependency is required.

Load some data

This example loads an indoor temperature and humidity dataset using the `TimeSeries.from_csv()` initializator. The time zone is assumed as UTC since not otherwise specified. The time series is then printed and plotted, with an (automatic) aggregation factor of ten to speed up the plotting and preventing web browsers to crash if using interactive plotting.

```
from timeseria import TEST_DATASETS_PATH as PATH
from timeseria.datastructures import TimeSeries

timeseries = TimeSeries.from_csv(PATH + "humitemp.csv")
```

[6] https://github.com/sarusso/Timeseria-notebooks
[7] https://hub.docker.com/r/sarusso/timeseria

```
Time series of #14000 points at variable resolution (~615s), from point @
1546475294.0 (2019- 01-03 00:28:14 +00:00) to point @ 1555544819.0 (2019
-04-17 23:46:59 +00:00)
```

```
timeseries.plot()
```

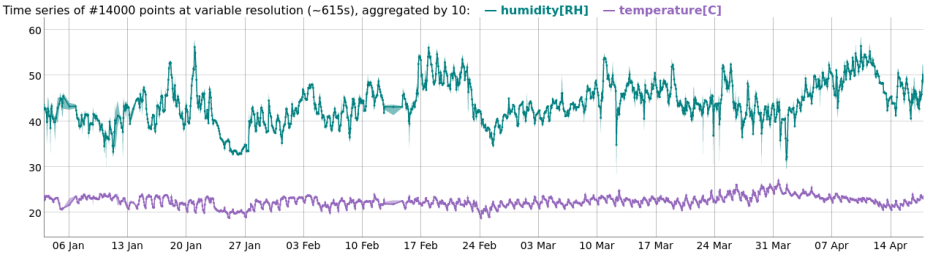


Figure 5.4: The time series plotted, with an (automatic) aggregator factor of ten. The area chart underlying the line chart indicates minimum and maximum values for each aggregated data point, in order to retain information about peaks.

Resample to hourly data

This example resamples the temperature time series loaded in the previous example at one hour sampling interval, making data uniform and equally spaced over time. Gaps are filled by linear interpolation (the default interpolation method when resampling) and the `data_loss` index is added to the data indexes. Then, it prints and plots the time series.

```
timeseries = timeseries.resample("1h")
print(timeseries)
```

```
[INFO] Using auto-detected sampling interval: 615.0s
[INFO] Resampled 14000 DataTimePoints in 2519 DataTimePoints

Time series of #2519 points at 1h resolution, from point @ 1546477200.0 (2019
-01-03 01:00:00 +00:00) to point @ 1555542000.0 (2019-04-17 23:00:00 +00:00)
```

```
timeseries.plot()
```

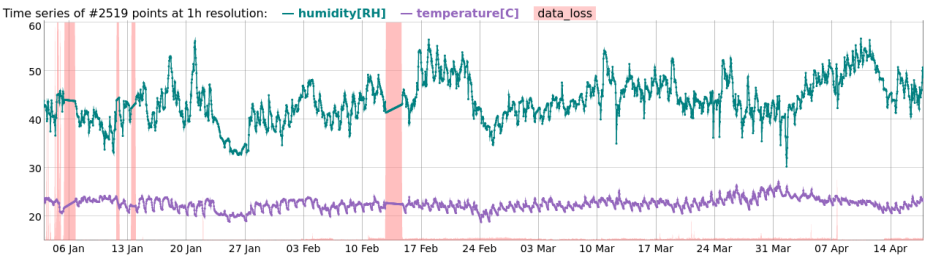


Figure 5.5: The resampled time series plotted. The data loss index is rendered as an overlapped red area chart.

Three-day hourly forecast

Using the previously resampled time series, this example first cross validates a LSTM neural network forecasting model with three rounds, meaning that the time series will be divided in six different datasets (three for fitting, three for evaluating). Then, it fits and applies such model with a forecasting horizon of 72 (hourly) steps, in order to get three days of temperature forecast. Data marked as missing is excluded from both the training and the evaluation of the model. The resulting time series is then plotted.

```
from timeseria.models import LSTMForecaster

forecaster = LSTMForecaster(window=24, neurons=256,
    ↪ features=["values", "hours"])

print(forecaster.cross_validate(timeseries, rounds=3,
    ↪ evaluate_error_metrics=["RMSE", "MAPE"]))

forecaster.fit(timeseries, epochs=100, reproducible=True)

timeseries = forecaster.apply(timeseries, steps=72).plot()
```

```
[INFO] Cross validation round 1/3: validate from 1546477200.0 (2019-01-03
01:00:00+00:00) to 1549497600.0 (2019-02-07 00:00:00+00:00), fit on the rest.
[INFO] Cross validation round 2/3: validate from 1549497600.0 (2019-02-07
00:00:00+00:00) to 1552518000.0 (2019-03-13 23:00:00+00:00), fit on the rest.
[INFO] Cross validation round 3/3: validate from 1552518000.0 (2019-03-13
23:00:00+00:00) to 1555538400.0 (2019-04-17 22:00:00+00:00), fit on the rest.

{'humidity[RH]_RMSE_avg': 1.168,
 'humidity[RH]_RMSE_stdev': 0.207,
 'humidity[RH]_MAPE_avg': 0.0196,
 'humidity[RH]_MAPE_stdev': 0.004,
 'temperature[C]_RMSE_avg': 0.362,
 'temperature[C]_RMSE_stdev': 0.039,
 'temperature[C]_MAPE_avg': 0.011,
 'temperature[C]_MAPE_stdev': 0.002}
```

```
timeseries.plot()
```

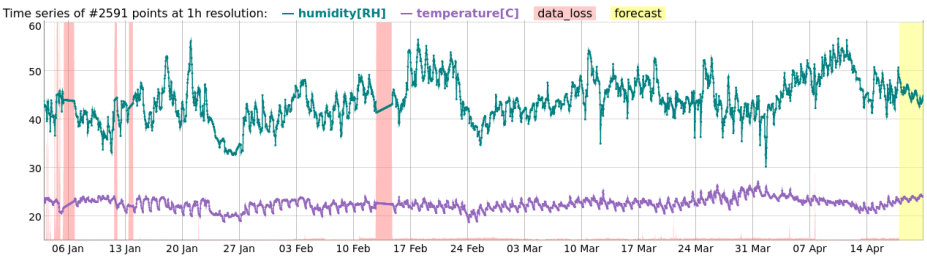


Figure 5.6: The time series plotted together with the forecast. The forecast is visible through its data index, rendered as a yellow area chart.

Perform anomaly detection

This example fits and applies a simple anomaly detection model based on periodic averages. By default the anomaly detectors assumes to work in unsupervised mode, using some "sane defaults". Data marked as missing is excluded from the anomaly detection process, in order to prevent false positives. The time series is then plotted together with the `anomaly` data index, computed as per Section 4.2, which allows for visual inspection: the bigger anomaly is detected around the 25th of January, when there is an unexpected drop with respect to the "normal" behavior.

```
from timeseria.models import PeriodicAverageAnomalyDetector

anomaly_detector = PeriodicAverageAnomalyDetector()

anomaly_detector.fit(time_series)

timeseries = anomaly_detector.apply(timeseries)
```

```
[INFO] Using a window of "24" for "humidity[RH]"
[INFO] Using a window of "24" for "temperature[C]"
[INFO] Predictive model(s) fitted, now evaluating...
[INFO] Computing actual vs predicted for "humidity[RH]"...
[INFO] Computing actual vs predicted for "temperature[C]"...
[INFO] Model(s) evaluated, now computing the error_metric distribution(s)...
[INFO] Anomaly detector fitted
```

```
timeseries.plot()
```

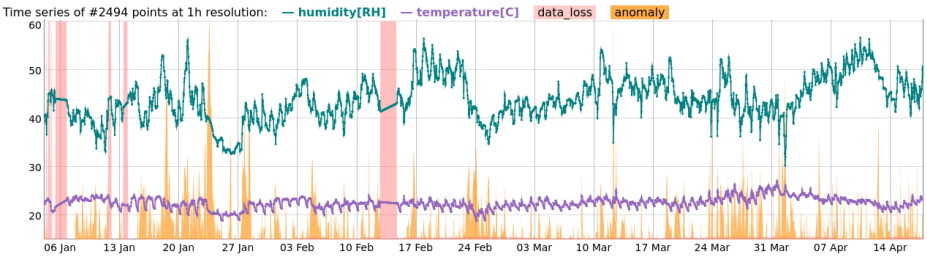


Figure 5.7: The time series plotted together with the anomaly index, rendered as an orange area chart.

Aggregate to daily data

This example aggregates the time series used in the previous examples into 1-day slots, also computing the minimum and maximum operations besides the (default) average one. Before aggregating, the time zone is changed to the right one for this dataset, in order to get the daily aggregates right, and to properly take into account the DST change in March. Data indexes are correctly recomputed and carried on in the aggregation process. The aggregated time series is then plotted, together with all of its (aggregated) data indexes.

```
timeseries.change_tz("Europe/Rome")
timeseries = timeseries.aggregate("1D",
    ↪ operations=["min","max","avg"]).plot()
```

```
[INFO] Using auto-detected sampling interval: 3600.0s
[INFO] Aggregated 2494 points in 103 slots
```

```
timeseries.plot()
```

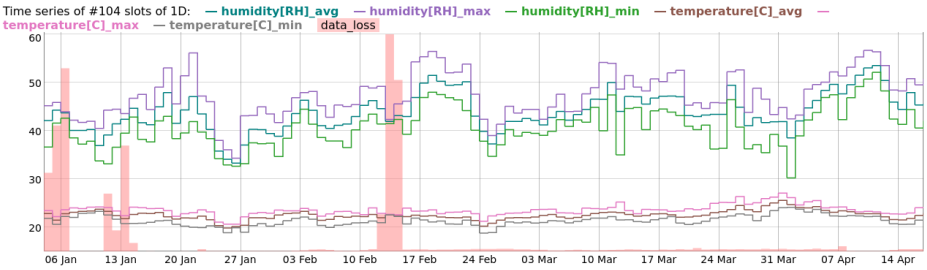


Figure 5.8: The aggregated time series plotted, as a step plot. To be noted that the data loss index was recomputed, according to the aggregation unit, and brought forward.

Chapter 6

A case study: Water Distribution Systems

6.1 Introduction

Water Distributions Systems (WDS) are a core part of modern city infrastructures, providing a constant water flow from facilities as treatment plants and wells to the final users. Over this path, many components are involved and subject to potential failures, as pipes, valves, pumps, storage tanks, and more. Leakages and breakages are the two most common type of failures, the second being a direct cause of the first one, if small enough not to be disruptive.

A leakage in an underground pipe that goes unnoticed does not only mean wasting a precious resource as water is, but can also cause erosion to the point of creating voids in the above terrain. If such erosion occur in a urban context, roads can collapse and building foundations can shift, posing a severe risk to the population. In case of sudden breakages instead, part of the buildings in the surroundings can be left without water, which is particularly problematic should one of these buildings be a critical infrastructure itself, as an hospital.

Over the course of recent years, ICT (Information and Communication Technologies) have seen an increasing adoption in the domain, not only in terms of software components, as Data Analytics (DA) tools and Decision Support Systems (DSS), but also as field-deployed solutions enabled by new hardware devices. More and more WDS are being equipped with remote monitoring systems, in order to better understand their dynamic, perform optimisations, and schedule maintenance tasks.

Some examples of these applications include burst detection [77, 12, 27, 13], water demand reconstruction [28] and forecasting [70, 18, 22], optimization of WDS segmentation [61, 29], water balances [59], evaluating key performance indicators [5] and developing new hydraulic and numerical models [24, 66, 89].

However, many issues can arise with the sensors themselves, in particular given the harsh environment in which they operate. Sensors not working properly can jeopardize such initiatives, and lead to unusable historical data.

It is therefore clear that having a robust anomaly detection process in place that operates in a timely manner can bring significant benefits, both in terms of detecting sensor issues and failures in the system itself, thus allowing to promptly address them.

In particular with respect to the the WDS domain, attempts of anomaly detection include Isolation Forest and K-Means [40], clustering and support vector regression [11], multi-class LDA [79], wavelet change-point detection methods [17] and logit models [37]. For anomaly classification, K-nearest neighbour [76], decision trees and SVM [93], and extracting short-term variations in combination with Isolation Forest [91] or Dynamic Type Warping [90, 30, 41].

Such attempts usually assume a static scenario, which is more oriented towards data mining applications rather than performing anomaly detection in an online fashion, which is a key factor for preventing issues in real-world scenarios.

Moreover, it has to be noted that the vast majority of the data used for scientific research in the WDS domain is from well known and curated datasets, as the Net3 dataset by EPANET or, most recently, simulation-based and synthetic data [58]. In particular, within the studies to be considered as relevant for this case-study, only a few tested their methods on real-world data [37, 76, 30, 4, 72], yet "tailoring" it for the specific task.

Given the global shift towards digitization [19] in the water management domain, we are entering a phase where new methodologies are not only required to be validated in an offline, synthetic environment, but also to be proven robust and scalable in real-world settings so that they can be transferred and deployed back to the industry.

The aim of this case-study is thus not only to asses the validity of the methodology proposed in this thesis work, but also to evaluate how it can be applied in such real-world scenarios.

6.2 Description

The dataset considered in this case study is from a WDS in the Friuli Venezia Giulia region, in northern Italy, which is currently monitored by Idrostudi s.r.l., an engineering company working in the water management domain.

Such dataset consists of 14 measurement points, which are locations of the WDS where one or more measurements are performed together. For each measurement point, flow rate and pressure sensor were deployed, and sampled each six minutes using a digital acquisition device. Flow rate is measured in l/s, while pressure in bar. The data data is then collected by a data logger with internet connectivity, and sent to a centralized server, in an ordered way. The total length of the data at the time of this work is of about two years, and Figure 6.1 shows an example of such data over a 7-day period.

As already mentioned, in real-world scenarios operating in harsh environments, it is common for the data acquisition process to fail. This can be due to hardware failures, lack of power or communication issues.

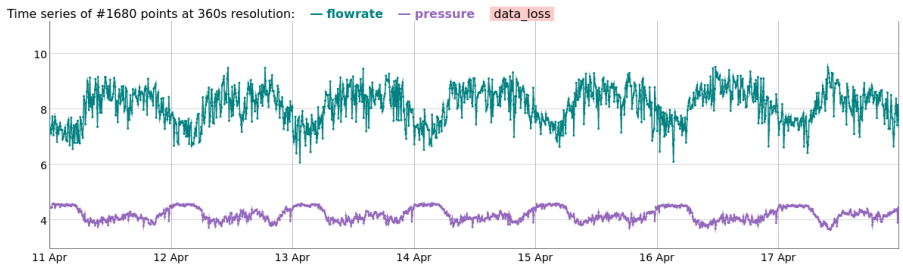


Figure 6.1: Example time series for one of the measurement points, over a 7-day period.

Hardware failures when reading sensors, if temporary, are often mitigated by retying multiple times to perform the measurement. However, when the retry limit is reached, or where there is just no such mechanism in places, it is common to see such value in the dataset to zero. This is perhaps not the best approach, as a "N.A." or null value would be far more suitable, but it is a well established practice in the monitoring hardware industry, including for the acquisition devices of this use-case, and it has therefore to be accounted for.

Losing data due to communication issues can be instead be almost completely avoided using local buffers. This does not mitigate the issue of loosing real-time, or near real-time access to the data, but when performing historical (or just delayed) analysis, such samples will be present. When instead a measurement could just not be performed, for example because the acquisition device was not operative (either due to a device-level failure or a lack of power), or because there was a transmission issues not handled by a buffer, then such measurements are just missing.

Image 6.2 shows an example of such hardware failures, when a data loss occurred. The missing data was detected and marked as such using the Timeseria library, which was used for all of the data pre-processing phase.

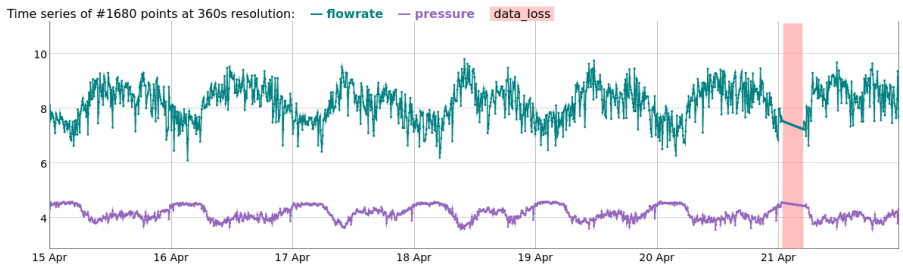


Figure 6.2: Example time series for one of the measurement points, over a 7-days period, and with a data loss (marked in red)

In general, data losses were frequent across the dataset, with an overall impact in the range of about 5-10%, as shown in Figure 6.3.

Figure 6.3: Example time series for one of the measurement points, over a 2-month period, with several data losses (marked in red)

6.3 Methods

6.3.1 Data pre-processing

The vast majority of data analysis an modeling techniques for time series data expect uniform, evenly spaces samples. In case of data losses, a common practice is to reconstruct the missing values, either via interpolation or more sophisticated techniques.

However, if by doing so the reconstructed values are indistinguishable form the others, then every subsequent step will be flawed: it is much different to draw conclusions when an analysis was performed on a time series with a 5% data loss than on a time series with a 47% data loss. The same applies for forecasting models, given that fitting a model on large portions of reconstructed data, or applying it using a window with a significant data loss, will cause the so called garbage-in garbage-out effect. It is therefore important to keep track of the data losses in the data, in particular when aiming for an anomaly detection process to be robust and capable of keeping under control the false positives.

Another key aspect of the data pre-processing is that it should be able to be applied both in an offline and in an online fashion. In particular with the latter, the resampling step must take into account whether it can create a new sample, or it has to wait for more "evidence". In other words, a data loss can be confirmed as such only when there is a "next" data point confirming that the data loss actually occurred, and that the acquisition process was not just delayed.

The Timeseria library was designed with these two requirements in mind: not only it attaches a "data loss" index to each data point when resampling, but it also does not declare any data loss unless it has evidence that it actually occurred, and it was thus used to resample the data.

Before resampling, undeclared failures in sensor readings had to be removed. As mentioned in the previous section, the acquisition devices in this use-case do not mark explicitly when a sensor could not be read, but they log a zero instead. Luckily, a sensors measuring a physical quantity will never measure exactly zero, due to the intrinsic noise in the measurement process. Such zero values were therefore just dropped before the resampling process, and consequently marked as data losses.

Once the data was resampled in this way, it become possible to exclude from the model fit phase the portions of the data presenting a data loss above a given threshold. This threshold was chosen to be equal to zero, that is, the models were fitted only on perfectly acquired data). In the inference phase, this threshold can (and has to) be relaxed, given that i) it has much smaller consequences to perform inference on potentially unreliable data rather that to use it to build that model, and ii) excluding from the anomaly detection process portions of the data just because of a small data loss would be counter-productive.

6.3.2 Model Selection

In model-based anomaly detection, any model capable of making a prediction is suitable for the task. Classical modeling approaches in the WDS domain make use of domain-specific numerical models as EPANET, which was for example used in [73] to detect leakages. Such models, although very accurate, require complex calibrations and are very intensive from a computational point of view.

Also taking a statistical approach could serve for the purpose, for example by computing the historical averages to then evaluate if the water consumption is not aligned with the historical behavior, as done in [46] for detecting failing or tampered meters. While such naive models can be useful to detect macroscopic issues to some extent, they are incapable of capturing the complex dynamics a WDS can be subject to and to perform fine-grained anomaly detection. ARIMA-like approaches (as SARIMA and ARIMAX) and derivatives (as Facebook Prophet) have to be excluded, as explained in section 4.4.

More sophisticated models have been implemented in recent years using Machine Learning (ML) and techniques. An example of such models, besides simple linear regression, is represented by SVM machines [**?**]. In the area of sewer systems, some studies already showed the superiority of ML over more classical approaches, as for example to to predict in-fluent flow rate [4].

The most promising approach to time series forecasting is based on Deep Learning (DL) techniques [74, 25]. However, in the field of Water Distribution Systems, adoption of DL is still limited. While some studies already evaluated simple feed-forward neural networks for water quality [72] and automated detection of pipe bursts [67], only recently research moved towards more complex architectures such as Convolutional and LSTM neural networks, as for example for urban water demand prediction [39], optimal pumps control [50], and overflow prediction in sewer systems [92, 44]. In all studies, result were very promising.

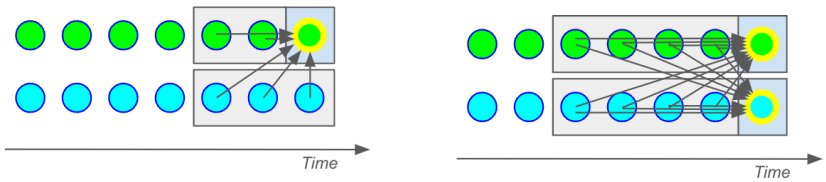
This thesis work will therefore build on top of these findings and make use of DL-based forecasting models. Within the models presented in section 4.3, the most accurate ones are the Transformer-based and the LSTM neural networks, while the others are either sub-optimal (as the RNNs) or focus on reducing the computational complexity (as the GRUs and TCNs), which is not relevant in this context: the datasets are not particularly large and there

are no requirements of running the anomaly detection on resource -constrained devices (in an edge computing fashion). For these reasons, only the LSTM and Transformer-based neural networks were considered in this use-case.

Moreover, an ensemble of two models was used, for each quantity: one more sensible to correlations, and another one more sensible to patterns, which are introduced below.

Correlation-based. This is a contextual forecasting model, as per section 3.2.2.2, with the aim of capturing the correlations between the physical quantities being monitored. Given a quantity to predict for the time step $t + 1$, the model has visibility over the other quantity (the context) for that timestamp as well, besides of n previous data points (the window size). This was set to 10 (one hour of data at 6-minute sampling intervals) in order to give context to the model without allowing it to capture any pattern. When an observation will be found not compatible with this model, and therefore the correlation breaks, it will likely be due to a sensor malfunction. Figure 6.4a summarize such model setup.

Pattern-based. This model is a pure forecasting model, and has visibly only of the previous n data points, with no information about the current time step. However, n (the window size) is sensibly longer than in the correlation-based model, enough to capture patterns, and thus set to 240 (24 hours of data at 6-minute sampling intervals). Such setup is summarized in Figure 6.4b. This model will thus be more likely to spot physical (dynamic) anomalies, where the correlation between the sensors might not break, but instead, intuitively speaking, all senors will agree that something "strange" is being measured.



(a) (b)

Figure 6.4: Schematic representation of correlation-based (a) and pattern-based (b) models setup. Each line of dots (green and blue) represents a physical quantity. The yellow highlight represents the prediction target (at $t + 1$).

In order to optimize the hyper parameters of these two sub-models, an evolutive algorithm was used, as described in Section 4.4.2, and in more in detail a relatively new approach: Deep Evolutionary Network Structured Representation (DENSER) [7].

This is a novel evolutionary approach that combines the principles of Genetic Algorithms (GAs) with those of Dynamic Structured Grammatical Evolution (DSGE) for automatically optimizing the hyperparameter of DL neural

networks, which was also released as an open source project under the name of Fast-DENSER [8].

Fast-DENSER in particularity is based on the concept of the *elite* evolution: for each population, the best individual (model) is marked as the elite and used as the basis for performing the mutations for the next generation.

The fitness function was designed to take into account the average error, the maximum error, and the goodness of the error distribution fit. Taking into account the maximum error allows to avoid choosing models that might be slightly better than others on average but that produce much bigger errors when they fail to make accurate predictions. This is particularly important for setting the anomaly index lower boundary, given that, as per Section 4.4, in semi-supervised mode the maximum error on the dataset used to define normality is still to be considered as normal.

The error metrics were chose as relative metrics, and in particular the MAPE and MaxAPE, while to assess the quality of the error distribution fit, the AIC was used, given the considerations of Section 4.2.2 about the p-value.

The overall fitness function of equation 4.22 was then adopted, with an AIC reference vale of 500, which was empirically found in terms of the average value assumed by the AIC:

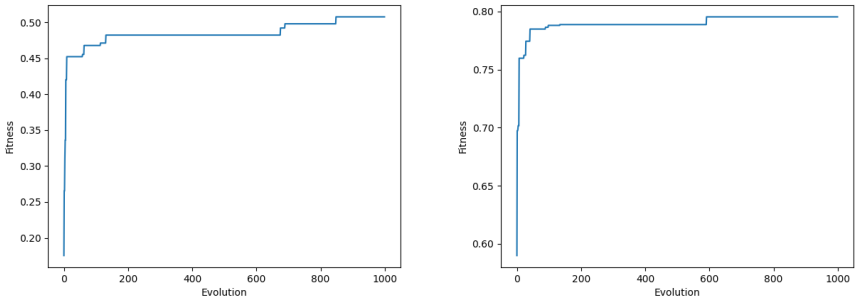
$$fitness = \frac{(1 - MAPE) + (1 - MaxAPE) + (1 - AIC/500)}{3} \tag{6.1}$$

A modified version of the Fast-DENSER code to support regressive tasks was used, together with two specific grammars: one for the Transformer-based architecture, and the other for the LSTM architecture.

Mutation rates where left as per default settings, and models were trained with a validation split of 0.9, for at most 100 epochs, with an early stopping mechanism in place which after 5 epochs of no sensible improvements stopped the training.

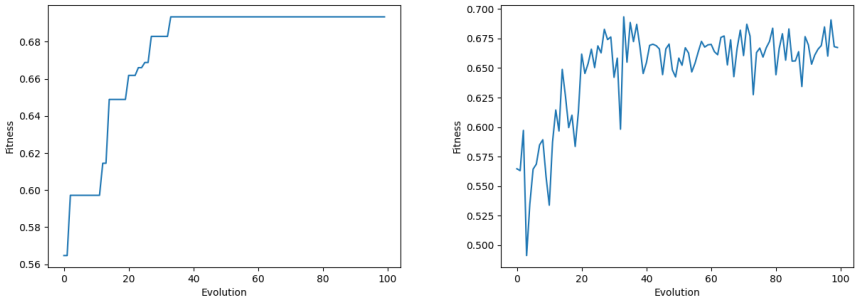
In order to evaluate for how many generations to run the evolutive process, a long run was performed for 1000 generations. Figure 6.5 shows the evolution of the fitness of the global elite (the best overall individual). Based on such experiments, the number of generations was set to 100.

Then, the evolutive runs were performed for each model and for each quantity. Figure 6.6 shows an example evolution with both global and local elite fitness, where for local is to be intended the elite of each generation, while Figure 6.7 shows the error distribution for an individual with a poor fitness (a) and for an individual with a good one (b).



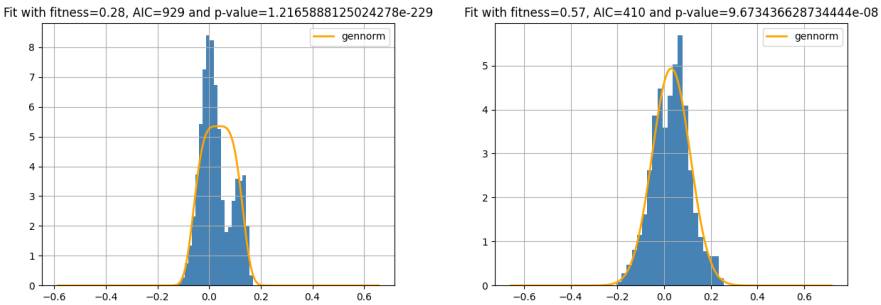
(a) (b)

Figure 6.5: Evolution over 1000 generations of the correlation-based (a) and pattern-based (b) models global elite fitness for the LSTM architecture.



(a) (b)

Figure 6.6: Evolution over 100 generations of the LSTM correlation-based model for pressure: global elite fitness (a) and local elite fitness (b).



(a) (b)

Figure 6.7: Examples error distributions for bad (a) and good (b) fitness individuals.

The dataset used for these experiments is a representative subset of the overall dataset, consisting in a time series of 10000 data points (Figure 6.8).

The simultaneous spikes in flow rate and drops in pressure that can be noted last for few tens of minutes and often indicate the presence of a high-demand events, such as the activation of an hydrant, which are part of the normal operation of the network and that will be thus required to be learned as much as possible..

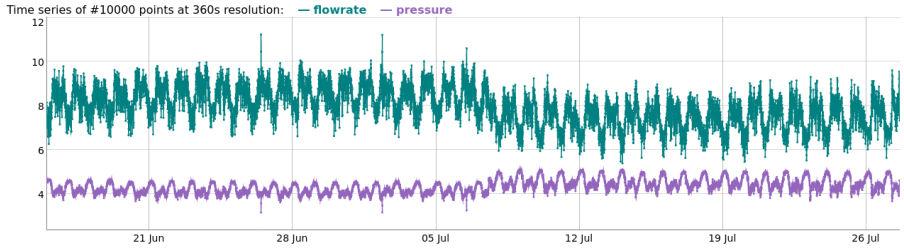


Figure 6.8: Representative data used for the evolutionary algorithm.

The results of the evolutionary algorithms are reported in Tables 6.1 and 6.2. Given the no clear winner, and since LSTM neural networks were 3 to 5 times faster to train with respect to Transformer-based one, and are simpler to handle, they were selected the reference architectures.

Quantity	**Architecture**	**Fitness**
pressure	LSTM	0.82
pressure	Transformer	0.86
flowrate	LSTM	0.85
flowrate	Transformer	0.70

Table 6.1: Evolutionary fitness results for the correlation-based models.

Quantity	**Architecture**	**Fitness**
pressure	LSTM	0.81
pressure	Transformer	0.83
flowrate	LSTM	0.82
flowrate	Transformer	0.79

Table 6.2: Evolutionary fitness results for the patter-based models.

6.3.3 Anomaly index settings

The anomaly index lower and upper boundaries were set based on considerations of Section 4.2.2. The lower bound was set to the maximum error made by the model on the dataset used as reference for normality, and as upper bound it was used an adherence corresponding to $1/10^{20}$.

The anomaly detection process described so far assigns a separate anomaly index to each quantity, and for each model. However, inspecting separately every single anomaly index of each quantity might be unpractical. One of the main advantages of the anomaly index defined in this thesis work is to allow merging anomaly indexes generated by different models on different quantities.

In general, a maximum operation will mark the whole data point with the maximum anomaly index found within its associated anomaly indexes, and tend to minimize the false negatives. Using the average (or even the minimum) will instead emphasize situations when most of the models "agreed", and would thus tend to lower or minimize the false positives.

Besides such considerations on false positives and negatives, which might not necessarily fit the use case, one could also make other, more specific arguments when combining different anomaly indexes. For example, if explicitly looking only for anomalies involving all the quantities (e.g. macroscopic failures), then combining different anomaly indexes with a minimum operation will not be related to minimizing the false negatives, it will just be the right operation by design.

In this use case, the interest is not only on all the models agreeing, or on just macroscopic failures, and thus the maximum operation was used to combine the different anomaly indexes, while keeping track of the original ones in order to allow for breaking down the various anomaly index when inspecting results.

6.3.4 Marking anomalous events

In order to inspect anomalies as events rather than as punctual observations, a simple strategy was employed to mark them as such: as an *anomaly event* is defined as any sequence of data points presenting an anomaly index greater than zero (the threshold was actually set 0.01 to allow for some tolerance) and lasting for at least two consecutive data points.

This choice was made to prevent isolated anomalies to trigger any alert, given that domain experts are much more interested in persistent anomalies, and given that it can greatly help in reducing false positives. It has to be noted that this approach introduces an intrinsic delay of one data point, i.e. after a data point presents an anomaly index greater than zero, it will be always necessary to wait for the next one before claiming an anomaly event and thus triggering an alert.

Gaps in anomaly events were also allowed, for a maximum of two data points, meaning that in case of less than two consecutive data points presenting an anomaly index of zero between one event and another, they will be merged together (and no alarm re-triggered).

For each anomaly event, the maximum anomaly index together with its average were also computed, which can be used to drawing conclusions about the magnitude and the "certainty level" of the event.

6.4 Results

Model architectures and hyperparameters were configured based on the findings of the evolutionary algorithms as per Section 6.3.2, and then plugged into a standard Timeseria forecaster, that was in turn used as the model-based anomaly detection engine. The data considered as reference for normality was the first month for each measurement point, validated by domain experts to include only nominal dynamics of the WDS.

In order to evaluate the forecasting models, the typical splitting between training, validation and test data is not possible in this context, given that it could end up omitting from the training some patterns still to be considered as normal, thus misleading the evaluation: all of the "normal" data has to be used for model training. To ensure that no over-fitting was occurring, a cross validation has been run on the same data, which showed no considerable differences between the cross validation rounds accuracies and the accuracy computed by training and testing the model on the entire "normal" dataset.

Tables 6.3 and 6.3 report the MAE and MaxAPE error metrics for the first four measurement points, while Figures 6.9 and 6.10 show an example of the predictions, for each time step.

MP	**MAPE**	**MaxAPE**
#1	Flow rate: 3% Pressure: 1%	Flow rate: 13% Pressure: 13%
#2	Flow rate: 12% Pressure: 1%	Flow rate: 54% Pressure: 6%
#3	Flow rate: 3% Pressure: 1%	Flow rate: 24% Pressure: 7%
#4	Flow rate: 6% Pressure: 2%	Flow rate: 58% Pressure: 20%

Table 6.3: Accuracies for the correlation-based model.

MP	MAPE	MaxAPE
#1	Flow rate: 4% Pressure: 1%	Flow rate: 28% Pressure: 14%
#2	Flow rate: 10% Pressure: 1%	Flow rate: 60% Pressure: 5%
#3	Flow rate: 3% Pressure: 1%	Flow rate: 26% Pressure: 6%
#4	Flow rate: 5% Pressure: 1%	Flow rate: 54% Pressure: 19%

Table 6.4: Accuracies for the pattern-based model.

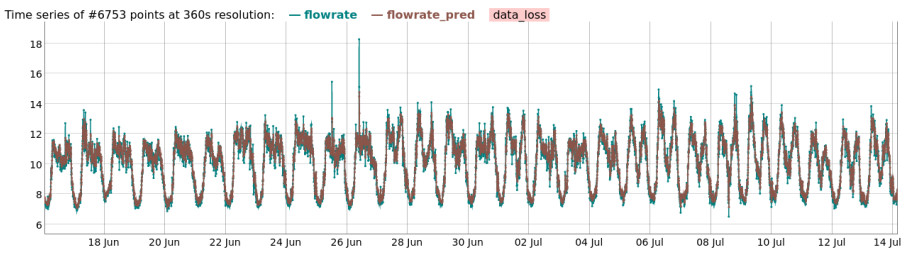


Figure 6.9: Example predictions for the correlation-based model on the flow rate quantity.

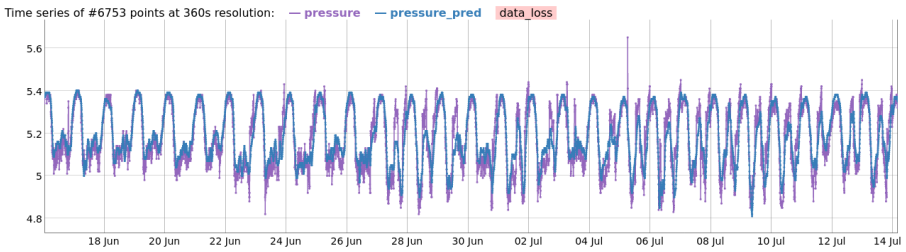


Figure 6.10: Example predictions for the correlation-based model on the pressure quantity.

Thanks to the Timeseria library, executing the anomaly detection process was then as simple as writing a Python code based on the following snippet, where the `get_keras_model()` support function is assumed to assemble the LSTM model based on the architecture and hyperparameters found by the evolutionary optimization:

```
def anomaly_detection(series_fit, series_inference, model_type):

    # Set parameters
    window = {'correlation': 10,
              'pattern': 240}
    features = {'correlation': ['values'],
                'pattern': ['values', 'hours'] }
    with_context = {'correlation': True,
                    'pattern': False}

    # Initialize and fit the forecasting models
    forecasters = {}
    for data_label in ['pressure', 'flowrate']:
        forecaster = LSTMForecaster(window=window[model_type],
            ↪ features=features[model_type],
            ↪ keras_model=get_keras_model(model_type, data_label))

        forecaster.fit(series_fit, reproducible=True, epochs=100,
            ↪ early_stopping=True, early_stopping_patience=5,
            ↪ normalize=True, normalization='max',
            ↪ with_context=with_context[model_type],
            ↪ target=data_label, verbose=True )

        forecasters[data_label] = forecaster

    # Initialize and fit the anomaly detector with the forecasters
    anomaly_detector =
        ↪ ModelBasedAnomalyDetector(models=forecasters)
    anomaly_detector.fit(series_fit,
        ↪ with_context=with_context[model_type],
        ↪ error_metric='PE', error_distribution='gennorm',
        ↪ verbose=True)

    # Perform the anomaly detection
    results_series = anomaly_detector.apply(series_inference,
        ↪ index_bounds=['max_err', 'adherence/10e20'],
        ↪ multivariate_index_strategy='max', verbose=True)

    return results_series
```

To be noted that such snippet can be easily modified to run the anomaly detection in an online fashion, by dropping the fit part and instead loading the anomaly detectors with `ModelBasedAnomlayDetector.load()`, provided that they where first saved somewhere after the fit phase. In this case, the `series_inference` would be the series corresponding to the model window.

Given the intrinsic impossibility of validating the anomalies due to the lack of a ground truth, anomaly detection results are reported mainly as qualitative observations. More in detail, anomalies were divided the anomalies in two main classes: trivial and nontrivial, for each of which some examples are reported.

Trivial anomalies

Such anomalies are trivial in the sense that they could have been probably spotted with simpler and more classical methods than DL. However, model-based anomaly detection, and in particular using DL, allows to detect them without manually setting any priors. Moreover, thanks to the anomaly index, an estimate about how "certain" such anomalies are is also available. In particular, trivial anomalies all show a maximum anomaly index of 1, meaning that at least a portion in each of them was considered as certainly anomalous by the proposed methodology. Some some examples with their descriptions are provided in Figures 6.11, 6.12 and 6.13.

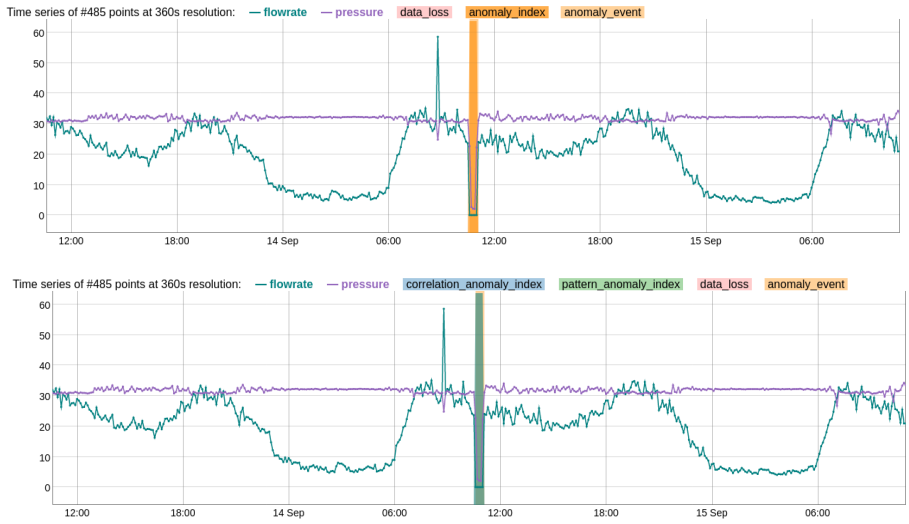


Figure 6.11: Example trivial anomaly #1. Combined (top), and with breakdown (bottom). Pressure rescaled for readability

In Figure 6.11, the anomaly is due to a sensor issue. Both quantities do not reach exactly zero, but they stop slightly above. This event has a maximum anomaly index of 1 and an average anomaly index of 1 as well: it is a certain anomaly. Both the correlation-based and the pattern-based models equally detected it. To be noted that the peak slight before this event has not been marked as anomalous, even if it deviates from the baseline of about the same, probably because i) such peaks were present in the training data as well and ii) the correlation does not break.

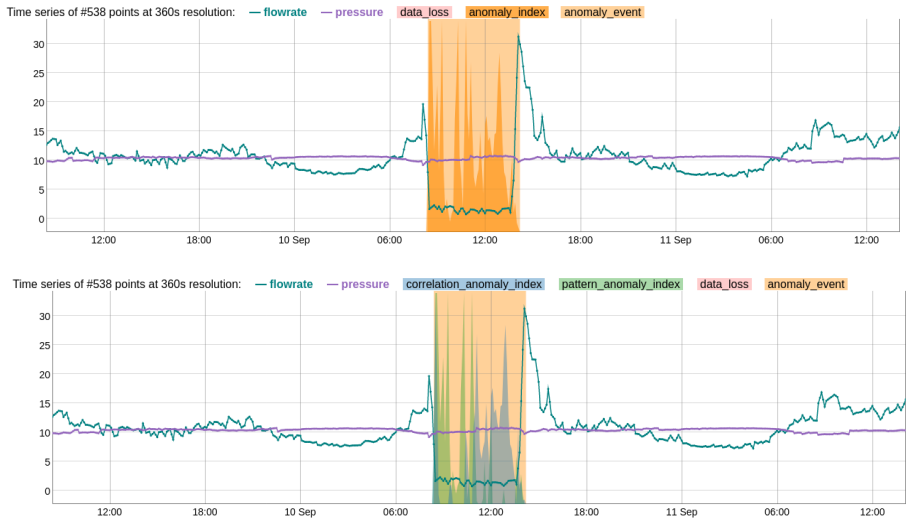


Figure 6.12: Example trivial anomaly #3. Combined (top), and with breakdown (bottom). Pressure rescaled for readability

In Figure 6.12, the anomaly is instead due to a temporary closure of a part of the WDS due to repair work. Interestingly, the first half was spotted by the pattern-based model, that then "adapted" to the low flow rate value, while the second half was spotted by the correlation-based model, probably due to the changed dynamic between the two quantities. This event a has a maximum anomaly index of 1 and an average anomaly index of 0.42: some parts of the event were certainly anomalous, but not all of it.

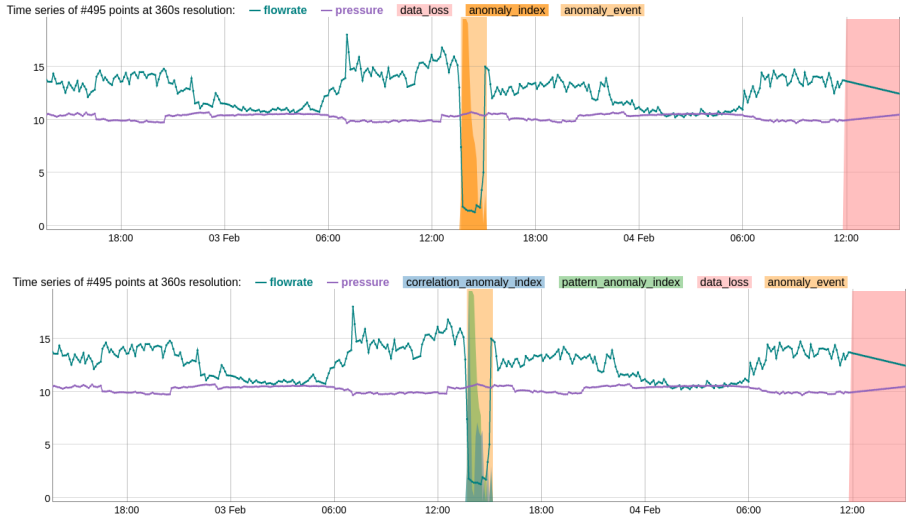


Figure 6.13: Example trivial anomaly #3. Combined (top), and with breakdown (bottom). Pressure rescaled for readability

In Figure 6.13, the anomaly is instead not (yet) explained, and it is currently under investigation by domain experts. It has has a maximum anomaly index of 1 and an average anomaly index of 0.42. It was mainly spot by the pattern-based model, which makes it a suspect malfunction or unreported closure of this section of the WDS.

Nontrivial anomalies

Nontrivial anomalies are more subtle, complex anomalies usually showing a maximum anomaly index lower than 1. In order to spot them without largely increase false positives rates, more sophisticated modeling is required, and is thus where model-based anomaly detection using DL can show its advantages. Nontrivial anomalies are often early-warning signs of a macroscopic anomaly on the way, and for this reason they are very precious: spotting them allows to react proactively. Example of such (rare) anomalies are provided in Figures 6.14 and 6.15, while in Figure 6.13 it is show how a portion of a time series suffering from a light drift is not erroneously marked as anomalous, thus showing that the propose methodology is proving robust in such circumstances.

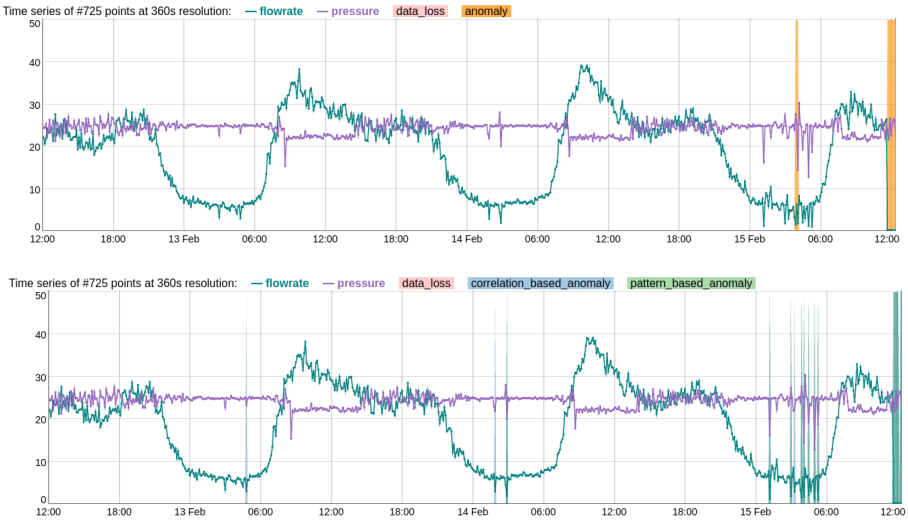


Figure 6.14: Example nontrivial anomaly #1. Combined (top), and with breakdown (bottom). Pressure rescaled for readability

In Figure 6.14, the nontrivial anomaly is the first one of the two, which corresponds to an increase in the noise picked up by the sensors at night (the lower parts of the spectrum). The maximum anomaly index of this event was 0.88, while the average 0.75. It was then followed by the second (trivial) anomaly, where the flow sensor The sensor then started to macroscopically malfunction and had to be replaced. It has to be noted that there were even more preliminary signals: single-data point anomalies were correctly detected even two days earlier. However, since at last two consecutive anomalous data points were required to mark an anomaly event, they would not have been

enough to trigger an alert.

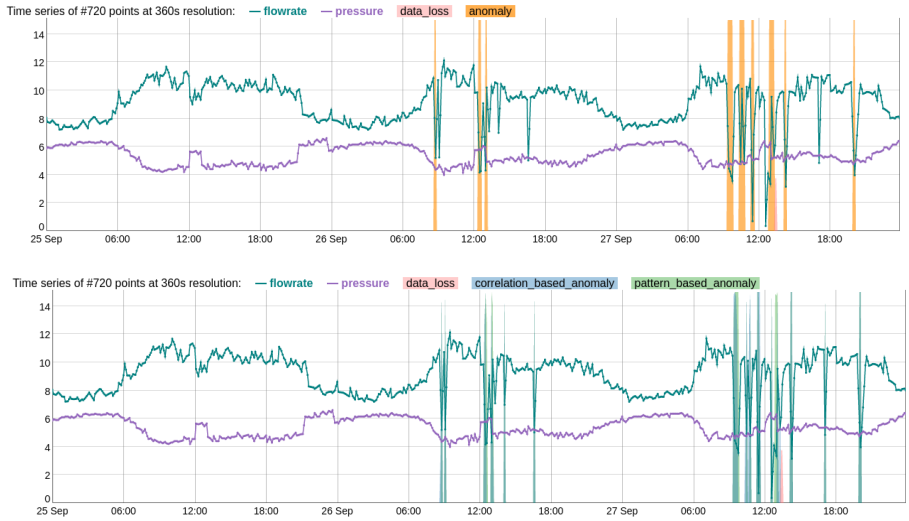


Figure 6.15: Example nontrivial anomaly #2. Combined (top), and with breakdown (bottom). Pressure rescaled for readability

In Figure 6.15, the anomaly is again due to a sensor malfunction, that started to have sudden drops in the flow. The values reached by the drops were present also in the data used as reference for the model training, but in this case they are much more prominent and were indeed were a symptom of a failing sensor, which then malfunctioned for months.

Lastly, Figure 6.16 shows an example of a pattern drift correctly not marked as anomalous. This data belong to the same measurement point of the data of the nontrivial anomaly reported in Figure 6.14, and was processed by the very same anomaly detection model. This shows that, while it might seem that the anomaly reported in Figure 6.14 was only detected because of the values in the lower part of the spectrum approaching zero, it was actually detected because of the variations of such values. Figure 6.16 shows indeed that, after the drift, values in the lower part approach (and reach) similar values as in Figure 6.14, but nothing is marked as anomalous.

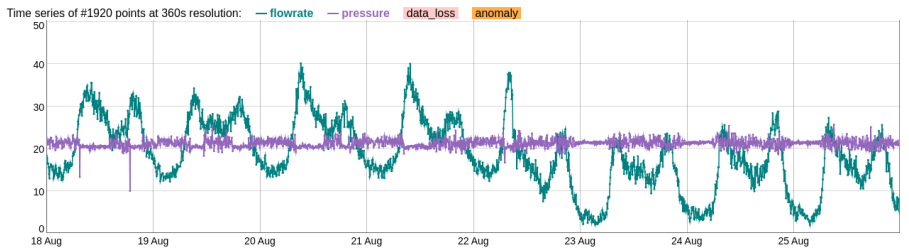


Figure 6.16: Example of a pattern drift correctly not marked as anomalous.

Chapter 7

Conclusions

Detecting anomalies is not a simple task. It is an ill-defined problem that spans several aspects, from technical challenges trough domain experience to deep philosophical questions as "what is normality"?

There is no ground truth about anomalies, which to be rightfully classified as such require to be truly new, unseen events: to some extent, detecting anomalies requires to quantify the unknown. For this reason, testing anomaly detection methods on "known" anomalies, even if using one of the few benchmarking datasets reasonably well-designed, risks to introduce bias, over-fitting and over-tuning, while providing limited to no informative power about how a given methodology will perform in front of real-world, new anomalies.

In the context of critical systems, where anomaly alerts can trigger high-priority operator responses or investigation, this is a major issue that has to be accounted for. Providing clear and reliable indications about the potential anomalies and their severity to the operators is indeed one of the main challenges, together with performing the detection in a timely manner, usually in real-time or near real-time.

Moreover, sensor-based critical systems often operate in harsh conditions and under strict operational requirements, which raise extra challenges with respect to the data quality, the resilience required by the anomaly detection processes, and the robustness of the algorithms.

This thesis work aimed at addressing such challenges by first questioning the very meaning of what anomalies represent, and then proposing a methodological framework to perform robust anomaly detection in this context.

Such framework was implemented in a novel time series processing library: Timeseria. Developed in Python following an object-oriented paradigm and a set of software development best practices as modularization, unit testing and containerization to ensure portability and reproducibility, Timeseria was designed to make it easier to manipulate time series data and to build statistical and machine learning models on top of it, ensuring a high level of consistency.

Then, the proposed methodological framework was tested on a real real-world case study in the Water Distribution Systems (WDS) domain. As a core part of modern city infrastructures, WDS provide constant water flow from

facilities like treatment plants and wells to the final users, and involve many components that are subject to potential failures, such as pipes, valves, pumps, storage tanks, and more.

A leakage in a WDS underground pipe that goes unnoticed does not only means wasting a precious resource as water is, but can also cause erosion to the point of creating voids in the above terrain. If such erosion occur in a urban context, roads can collapse and building foundations can shift, posing a severe risk to the population. In case of sudden breakages instead, part of the buildings in the surroundings can be left without water, which is particularly problematic should one of these buildings be a critical infrastructure itself, as an hospital.

The case study considered in this thesis work consisted of 14 nodes of a WDS in the Friuli Venezia Giulia region, in northern Italy. The dataset comprised flow rate and pressure sensor measurements for each node, sampled at six-minute intervals. The dataset showed several data losses, all detected and marked as such with the Timeseria library, which was used for both data pre-processing and to perform the anomaly detection using a model-based approach.

Several models have been considered, and the choice fell on a set of deep learning models, because of their ability to capture complex nonlinear dependencies. Within the deep learning models suitable for time series forecasting, the best compromise was to use a Long Short Term Memory (LSTM) neural network. In order to capture both the intrinsic correlation between the pressure and the flow rate and their typical patterns in a separate way, an ensemble of two types of models has been used. The first was trained to predict the value for the next time step given only a window of past data (the previous 24 hours), in order to capture the patterns. The second was instead trained in order to capture the correlations. More in detail, such model was trained to predict the value of a target quantity (e.g. flow rate) at the next step given the contextual data at the same next time step (e.g. pressure). It also used a much smaller window of past data (of 1 hour) to give some context to the model but not enough to capture any pattern.

This case study showed how the methodology proposed in this thesis work can be applied in a real-world scenario in order to perform robust anomaly detection in the context of sensor-based critical systems, in an end-to-end fashion. The anomaly index defined according to the proposed methodology allowed differentiating the anomalies based on their magnitude, thus providing a clear indication about their severity.

The results provided (besides model accuracies) are mainly qualitative, given the lack of ground truth about anomalies, and reported that trivial anomalies (as planned pipe closures) could all be successfully identified with great confidence, while some more subtle and nontrivial anomalies, often symptoms of upcoming issues as sensor failures, could be identified as well.

Lastly, no significant false positives were found: all of the detected anomalies were of interest for the domain experts involved in the process, which was one of

the main goals of this thesis work. This was made possible by making the proposed methodology robust by design: despite the several data losses, they were all correctly handled both in the model training and inference phases, without capturing distorted patterns or performing unreliable predictions. Moreover, optimizing the models not only for the average error but also for a contained maximum error, and targeting a convenient error distribution, allowed to pose a solid basis on which to calibrate the anomaly index. This in turn allowed to obtain clear and reliable indications about the absolute degree of suspicion of the potential anomalies.

Future directions include several paths. The model selection process can be improved, starting by tuning hyperparameters on a per-series basis rather that for the overall dataset. Also the error modeling can be improved, for example the Timeseria library could be modified in order to support multimodal or asymmetric error distributions, although the implications of accepting an asymmetric error have to be carefully considered.

The anomaly index could be further developed to work on joint error probabilities over a given prediction horizon rather than punctual ones, or to take into account the cumulative error, should the use-case allow for some lag in the anomaly detection.

Adding extra modeling (e.g. pattern recognition) for capturing "known" and potentially unpredictable rare events (as high-demand events in WDS) could help in improving the methodology even further, given that it could be used to exclude such events both when computing the error distribution of the models and in the detection phase. Also in this scenario the lag that would be introduced (a pattern requires to be formed before it can be detected) has to be taken into account, and its applicability thus depends on the type of patterns involved and on how stringent the "timely" requirement is. However, it looks like a promising path.

Lastly, the methodology proposed in this thesis work could be evolved to work with models capable of predicting probability distributions rather than punctual values, as Bayesian models, probabilistic neural networks, and generative models.

Bibliography

[1] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, 2017.

[2] Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C Maddix, Syama Sundar Rangapuram, David Salinas, Jasper Schulz, et al. Gluonts: Probabilistic and neural time series modeling in python. *J. Mach. Learn. Res.*, 21(116):1–6, 2020.

[3] Sarah Alnegheimish, Dongyu Liu, Carles Sala, Laure Berti-Equille, and Kalyan 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.

[4] Mozafar Ansari, Faridah Othman, Taher Abunama, and Ahmed El-Shafie. Analysing the accuracy of machine learning techniques to develop an integrated influent time series model: case study of a sewage treatment plant, malaysia. *Environmental Science and Pollution Research*, 25:1–11, 04 2018.

[5] ARERA. Annual report on the state of services and regulatory activity, 2021.

[6] Kevin Ashton et al. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[7] Filipe Assunção, Nuno Lourenço, Penousal Machado, and Bernardete Ribeiro. Denser: deep evolutionary network structured representation. *Genetic Programming and Evolvable Machines*, 20:5–35, 2019.

[8] Filipe Assunção, Nuno Lourenço, Bernardete Ribeiro, and Penousal Machado. Fast-denser: Fast deep evolutionary network structured representation. *SoftwareX*, 14:100694, 2021.

[9] Christopher M Bishop. Pattern recognition and machine learning. *Springer google schola*, 2:1122–1128, 2006.

[10] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A Lozano. A review on outlier/anomaly detection in time series data. *ACM computing surveys (CSUR)*, 54(3):1–33, 2021.

[11] Antonio Candelieri. Clustering and support vector regression for water demand forecasting and anomaly detection. *Water*, 9(3), 2017.

[12] Miguel Capelo, Bruno Brentan, Laura Monteiro, and Dídia Covas. Near-real time burst location and sizing in water distribution systems using artificial neural networks. *Water*, 13:1884, 07 2021.

[13] Caterina Capponi, Marco Ferrante, Aaron Zecchin, and James Jinzhe Gong. Leak detection in a branched system by inverse transient analysis with the admittance matrix method. *Water Resources Management*, 31:1–15, 10 2017.

[14] Tinkoff.ru Artificial Intelligence Center. Etna - predict your time series the easiest way. `https://github.com/tinkoff-ai/etna`. Accessed: 2024-05-12.

[15] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[16] Maximilian Christ. Standardize time series formats. `https://web.archive.org/web/20230124141947/https://github.com/MaxBenChrist/awesome_time_series_in_python/blob/master/standardize_time_series_formats.md`, 2021. Accessed: 2023-01-24.

[17] Symeon E. Christodoulou, Elena Kourti, and Agathoklis Agathokleous. Waterloss Detection in Water Distribution Networks using Wavelet Change-Point Detection. *Water Resources Management: An International Journal, Published for the European Water Resources Association (EWRA)*, 31(3):979–994, February 2017.

[18] Andrea Cominola, K. Nguyen, Matteo Giuliani, Rodney Stewart, Holger Maier, and Andrea Castelletti. Data mining to uncover heterogeneous water use behaviors from smart meter data. *Water Resources Research*, 55, 11 2019.

[19] European Commission, Content Directorate-General for Communications Networks, Technology, and G Anzaldi Varas. *Digital single market for water services action plan : final report*. Publications Office, 2020.

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

[21] Paul SP Cowpertwait and Andrew V Metcalfe. *Introductory time series with R*. Springer Science & Business Media, 2009.

[22] Enrico Creaco, Giacomo Galuppini, Alberto Campisano, and Marco Franchini. Bottom-up generation of peak demand scenarios in water distribution networks. *Sustainability*, 13(1):1–18, 2021.

[23] Axel Daneels and Wayne Salter. What is scada? 1999.

[24] Nhu Do, Angus Simpson, Jochen Deuerlein, and Olivier Piller. Calibration of water demand multipliers in water distribution systems using genetic algorithms. *Journal of Water Resources Planning and Management*, 142:04016044, 06 2016.

[25] Meftah Elsaraiti and Adel Merabet. A comparative analysis of the arima and lstm predictive models and their effectiveness for predicting wind speed. *Energies*, 14(20):6782, 2021.

[26] facebook Research. Kats - a one stop shop for time series analysis in python. `https://web.archive.org/web/20221004032627/https://github.com/facebookresearch/Kats`, 2021. Accessed: 2023-01-06.

[27] Zahra Fereidooni, Hooman Tahayori, and Ali Bahadori-Jahromi. A hybrid model-based method for leak detection in large scale water distribution networks. *Journal of Ambient Intelligence and Humanized Computing*, 12, 02 2021.

[28] Diana Fiorillo, Giacomo Galuppini, Enrico Creaco, F. Paola, and Maurizio Giugni. Identification of influential user locations for smart meter installation to reconstruct the urban demand pattern. *Journal of Water Resources Planning and Management*, 146:04020070, 06 2020.

[29] S. Fiorindo, L. Zovatto, L. Falcomer, Murari E., and Zanello F. Automatic optimization of the water supply networks segmentation. 2018.

[30] Caspar Geelen, Doekle Yntema, Jaap Molenaar, and Karel Keesman. Monitoring support for water distribution systems based on pressure sensor data. *Water Resources Management*, 33, 08 2019.

[31] Paul Goodwin and Richard Lawton. On the asymmetry of the symmetric mape. *International journal of forecasting*, 15(4):405–408, 1999.

[32] Frank E Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

[33] Douglas M Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[34] Julien Herzen, Francesco LÃ¤ssig, Samuele Giuliano Piazzetta, Thomas Neuer, LÃ©o Tafti, Guillaume Raille, Tomas Van Pottelbergh, Marek Pasieka, Andrzej Skrodzki, Nicolas Huguenin, Maxime Dumonal, Jan KoÅcisz, Dennis Bader, FrÃ©dÃ©rick Gusset, Mounir Benheddi, Camila Williamson, Michal Kosinski, Matej Petrik, and GaÃl Grosch. Darts: User-friendly modern machine learning for time series. *Journal of Machine Learning Research*, 23(124):1–6, 2022.

[35] Mike Hinchey and Lorcan Coyle. Evolving critical systems: A research agenda for computer-based systems. In *2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, pages 430–435. IEEE, 2010.

[36] Reza Hosseini, Albert Chen, Kaixu Yang, Sayan Patra, Yi Su, Saad Eddin Al Orjany, Sishi Tang, and Parvez Ahammad. Greykite: Deploying flexible forecasting at scale at linkedin. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3007–3017, 2022.

[37] Mashor Housh and Avi Ostfeld. An integrated logit model for contamination event detection in water distribution systems. *Water Research*, 75:210–223, 2015.

[38] Stephan Hoyer and Joe Hamman. xarray: Nd labeled arrays and datasets in python. *Journal of Open Research Software*, 5(1), 2017.

[39] Piao Hu, Jun Tong, Jingcheng Wang, Yue Yang, and Luca de Oliveira Turci. A hybrid model based on cnn and bi-lstm for urban water demand prediction. In *2019 IEEE Congress on evolutionary computation (CEC)*, pages 1088–1094. IEEE, 2019.

[40] Zukang Hu, Wenlong Chen, Helong Wang, Pei Tian, and Dingtao Shen. Integrated data-driven framework for anomaly detection and early warning in water distribution system. *Journal of Cleaner Production*, 373:133977, 2022.

[41] Pingjie Huang, Naifu Zhu, Dibo Hou, Jinyu Chen, Yao Xiao, Jie Yu, Guangxin Zhang, and Hongjian Zhang. Real-time burst detection in district metering areas in water distribution system based on patterns of water demand with supervised learning. *Water*, 10(12), 2018.

[42] Kyle Hundman, Valentino Constantinou, Christopher Laporte, Ian Colwell, and Tom Soderstrom. Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 387–395, 2018.

[43] Vincent Jacob, Fei Song, Arnaud Stiegler, Bijan Rad, Yanlei Diao, and Nesime Tatbul. Exathlon: A benchmark for explainable anomaly detection over time series. *arXiv preprint arXiv:2010.05073*, 2020.

[44] Hoon Kang, Seunghyeok Yang, Huang Jianying, and Jeill Oh. Time series prediction of wastewater flow rate by bidirectional lstm deep learning. *International Journal of Control, Automation and Systems*, 18:3023–3030, 12 2020.

[45] Eamonn Keogh, Jessica Lin, Sang-Hee Lee, and Helga Van Herle. Finding the most unusual time series subsequence: algorithms and applications. *Knowledge and Information Systems*, 11:1–27, 2007.

[46] Einat Kermany, Hanna Mazzawi, Dorit Baras, Yehuda Naveh, and Hagai Michaelis. Analysis of advanced meter infrastructure data of water consumption in apartment buildings. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 1159–1167, New York, NY, USA, 2013. Association for Computing Machinery.

[47] Deborah Khider, Feng Zhu, and Yolanda Gil. autots: Automated machine learning for time series analysis. In *AGU Fall Meeting Abstracts*, volume 2019, pages PP43D–1637, 2019.

[48] HuaMing Huang Kishan G. Mehrotra, Chilukuri K. Mohan. *Anomaly Detection Principles and Algorithms*. Springer, 2017.

[49] Krzysztof Kotowski, Christoph Haskamp, Jacek Andrzejewski, Bogdan Ruszczak, Jakub Nalepa, Daniel Lakey, Peter Collins, Aybike Kolmas, Mauro Bartesaghi, Jose Martinez-Heras, et al. European space agency benchmark for anomaly detection in satellite telemetry. *arXiv preprint arXiv:2406.17826*, 2024.

[50] Christian Kühnert, Naga Mamatha Gonuguntla, Helene Krieg, Dimitri Nowak, and Jorge A Thomas. Application of lstm networks for water demand prediction in optimal pump control. *Water*, 13(5):644, 2021.

[51] Daesoo Lee, Sara Malacarne, and Erlend Aune. Explainable time series anomaly detection using masked latent generative modeling. *Pattern Recognition*, 156:110826, 2024.

[52] Mingfeng Lin, Henry C Lucas Jr, and Galit Shmueli. Research commentary — too big to fail: large samples and the p-value problem. *Information systems research*, 24(4):906–917, 2013.

[53] Xing Liu, Cheng Qian, William Grant Hatcher, Hansong Xu, Weixian Liao, and Wei Yu. Secure internet of things (iot)-based smart-world critical infrastructures: Survey, case study and research opportunities. *IEEE Access*, 7:79523–79544, 2019.

[54] Markus Löning, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, and Franz J Király. sktime: A unified interface for machine learning with time series. *arXiv preprint arXiv:1909.07872*, 2019.

[55] Engineering Services LP. Deepwater horizon blowout preventer failure analysis report. `https://www.csb.gov/assets/1/20/appendix_2_a_deepwater_horizon_blowout_preventer_failure_analysis1.pdf`, 2014.

[56] Wes McKinney et al. pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9, 2011.

[57] Madeleine McPherson, Theofilos Sotiropoulos-Michalakakos, LD Danny Harvey, and Bryan Karney. An open-access web-based tool to access global, hourly wind and solar pv generation time-series derived from the merra reanalysis dataset. *Energies*, 10(7):1007, 2017.

[58] Andrea Menapace, Ariele Zanfei, Manuel Felicetti, Diego Avesani, Maurizio Righetti, and Rudy Gargano. Burst detection in water distribution systems: The issue of dataset collection. *Applied Sciences*, 10(22), 2020.

[59] Jordi Meseguer and Joseba Quevedo. *Real-Time Monitoring and Control in Water Systems*, pages 1–19. 05 2017.

[60] S. Amizadeh N. Laptev and Y. Billawala. A labeled anomaly detection dataset, version 1.0 (16m). `https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70`, 2015.

[61] Matteo Nicolini, Carlo Giacomello, and Kalyanmoy Deb. Calibration and optimal leakage management for a real water distribution network. *Journal of Water Resources Planning and Management*, 137, 01 2011.

[62] Malte Ollenschläger, Arne Küderle, Wolfgang Mehringer, Ann-Kristin Seifer, Jürgen Winkler, Heiko Gaßner, Felix Kluge, and Bjoern M Eskofier. Mad gui: An open-source python package for annotation and analysis of time-series data. *Sensors*, 22(15):5849, 2022.

[63] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)*, 54(2):1–38, 2021.

[64] John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S Tsay, Themis Palpanas, and Michael J Franklin. Tsb-uad: an end-to-end benchmark suite for univariate time-series anomaly detection. *Proceedings of the VLDB Endowment*, 15(8):1697–1711, 2022.

[65] Rocio Lopez Perez, Florian Adamsky, Ridha Soua, and Thomas Engel. Forget the myth of the air gap: Machine learning for reliable intrusion detection in scada systems. *EAI Endorsed Transactions on Security and Safety*, 6(19):e3–e3, 2019.

[66] Zhang Qingzhou, Feifei Zheng, Huan-Feng Duan, Yueyi Jia, Tuqiao Zhang, and Xinlei Guo. Efficient numerical approach for simultaneous calibration of pipe roughness coefficients and nodal demands for water distribution systems. *Journal of Water Resources Planning and Management*, 144:04018063, 07 2018.

[67] Michele Romano, Zoran Kapelan, and Dragan Savic. Automated detection of pipe bursts and other events in water distribution systems. *Journal of Water Resources Planning and Management*, 140:457–467, 04 2014.

[68] Karen Rose, Scott Eldridge, and Lyman Chapin. The internet of things: An overview. *The internet society (ISOC)*, 80(15):1–53, 2015.

[69] Bogdan Ruszczak, Krzysztof Kotowski, Jacek Andrzejewski, Christoph Haskamp, and Jakub Nalepa. Oxi: An online tool for visualization and annotation of satellite time series data. *SoftwareX*, 23:101476, 2023.

[70] Jordi Saludes, Joseba Quevedo, and Vicenç Puig. *Demand Forecasting for Real-Time Operational Control*, pages 99–111. 05 2017.

[71] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. Anomaly detection in time series: a comprehensive evaluation. *Proceedings of the VLDB Endowment*, 15(9):1779–1797, 2022.

[72] Lina Sela, Jonathan Arad, Mashor Housh, and Avi Ostfeld. Event detection in water distribution systems from multivariate water quality time series. *Environmental science technology*, 46:8212–9, 06 2012.

[73] Yu Shao, Xin Li, Tuqiao Zhang, Shipeng Chu, and Xiaowei Liu. Time-series-based leakage detection using multiple pressure sensors in water distribution systems. *Sensors*, 19(14), 2019.

[74] Sima Siami-Namini, Neda Tavakoli, and Akbar Siami Namin. A comparison of arima and lstm in forecasting time series. In *2018 17th IEEE international conference on machine learning and applications (ICMLA)*, pages 1394–1401. IEEE, 2018.

[75] Julien Siebert, Janek Groß, and Christof Schroth. A systematic review of packages for time series analysis. *Engineering Proceedings*, 5(1):22, 2021.

[76] Adrià Soldevila, Joaquim Blesa, Sebastian Tornil-Sin, Eric Duviella, Rosa M. Fernandez-Canti, and Vicenç Puig. Leak localization in water distribution networks using a mixed model-based/data-driven approach. *Control Engineering Practice*, 55:162–173, 2016.

[77] Sophocles Sophocleous, Dragan Savic, and Zoran Kapelan. Leak localization in a real water distribution network based on search-space reduction. *Journal of Water Resources Planning and Management*, 145, 04 2019.

[78] Ya Su, Youjian Zhao, Chenhao Niu, Rong Liu, Wei Sun, and Dan 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*, pages 2828–2837, 2019.

[79] Congcong Sun, Benjamí Parellada, Vicenç Puig, and Gabriela Cembrano. Leak localization in water distribution networks using pressure and data-driven classifier approach. *Water*, 12(1):54, Dec 2019.

[80] Romain Tavenard, Johann Faouzi, Gilles Vandewiele, Felix Divo, Guillaume Androz, Chester Holtz, Marie Payne, Roman Yurchak, Marc Rußwurm, Kushal Kolar, et al. Tslearn, a machine learning toolkit for time series data. *J. Mach. Learn. Res.*, 21(118):1–6, 2020.

[81] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[82] Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

[83] Jeroen Van Der Donckt, Jonas Van Der Donckt, Emiel Deprost, Nicolas Vandenbussche, Michael Rademaker, Gilles Vandewiele, and Sofie Van Hoecke. Do not sleep on traditional machine learning: Simple and interpretable techniques are competitive to deep learning for sleep scoring. *Biomedical Signal Processing and Control*, 81:104429, 2023.

[84] Jonas Van Der Donckt, Jeroen Van Der Donckt, Emiel Deprost, and Sofie Van Hoecke. tsflex: Flexible time series processing & feature extraction. *SoftwareX*, 17:100971, 2022.

[85] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[86] Dennis Wagner, Tobias Michels, Florian CF Schulz, Arjun Nair, Maja Rudolph, and Marius Kloft. Timesead: Benchmarking deep multivariate time-series anomaly detection. *Transactions on Machine Learning Research*, 2023.

[87] Phillip Wenig, Sebastian Schmidl, and Thorsten Papenbrock. Timeeval: A benchmarking toolkit for time series anomaly detection algorithms. *Proceedings of the VLDB Endowment*, 15(12):3678–3681, 2022.

[88] Renjie Wu and Eamonn 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, 2021.

[89] Zhou Xiao, Weirong Xu, Kunlun Xin, Hexiang Yan, and Tao Tao. Self-adaptive calibration of real-time demand and roughness of water distribution systems. *Water Resources Research*, 54, 07 2018.

[90] Lu Xing and Lina Sela. Unsteady pressure patterns discovery from high-frequency sensing in water distribution systems. *Water Research*, 158:291–300, 2019.

[91] Weirong Xu, Xiao Zhou, Kunlun Xin, Joby Boxall, Hexiang Yan, and Tao Tao. Disturbance extraction for burst detection in water distribution networks using pressure measurements. *Water Resources Research*, 56(5):e2019WR025526. e2019WR025526 2019WR025526.

[92] Duo Zhang, Geir Lindholm, and Harsha Ratnaweera. Use long short-term memory to enhance internet of things for combined sewer overflow monitoring. *Journal of Hydrology*, 556:409–418, 2018.

[93] Xiangqiu Zhang, Zhihong Long, Tian Yao, Hua Zhou, Tingchao Yu, and Yongchao Zhou. Real-time burst detection based on multiple features of pressure data. *Water Supply*, 22(2):1474–1491, 10 2021.

[94] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: A python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.