Classification of Anomalies in Telecommunication Network KPI Time Series

Korantin Bordeau–Aubert

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Computer Science (MCompSc) at

Concordia University

Montréal, Québec, Canada

September 2023

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Korantin Bordeau–Aubert**

Entitled: **Classification of Anomalies in Telecommunication Network KPI Time Series**

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science (MCompSc)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Mirco Ravanelli

_____ Examiner
Dr. Abdelhak Bentaleb

_____ Supervisor
Dr. Tristan Glatard

_____ Co-supervisor
Dr. Brigitte Jaumard

Approved by _____
Joey Paquet, Chair
Department of Computer Science

_____ 2023

_____
Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Classification of Anomalies in Telecommunication Network KPI Time Series

Korantin Bordeau–Aubert

The increasing complexity and scale of telecommunication networks have led to a growing interest in automated anomaly detection systems. However, the classification of anomalies detected on network Key Performance Indicators (KPI) has received less attention, resulting in a lack of information about anomaly characteristics and classification processes. To address this gap, this thesis proposes a modular anomaly classification framework. The framework assumes separate entities for the anomaly classifier and the detector, allowing for a distinct treatment of anomaly detection and classification tasks on time series. The objectives of this study are (1) to develop a time series simulator that generates synthetic time series resembling real-world network KPI behavior, (2) to build a detection model to identify anomalies in the time series, (3) to build classification models that accurately categorize detected anomalies into predefined classes (4) to evaluate the classification framework performance on simulated and real-world network KPI time series. This study has demonstrated the good performance of the anomaly classification models trained on simulated anomalies when applied to real-world network time series data.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my research supervisors Dr Tristan Glatard and Dr Brigitte Jaumard for their invaluable guidance and insights during this research. I am deeply indebted to Tristan for his counseling and supervision in the last months of the thesis, which greatly contributed to the research and writing progress. I am also extremely grateful to Brigitte for her active assistance and propositions provided during the first part of this thesis. Their expertise greatly helped me in shaping this research and writing this thesis. I extend my gratitude for your patience and unwavering support throughout my master's degree journey.

Additionally, I am sincerely grateful to Sylvain Nadeau, Director - Strategic Innovation Edge, EXFO Inc., for their generous financial and technical support, which generously provided me great insight and knowledge about network KPI and applications. This project wouldn't have been possible without the resources, guidance, and comments he provided.

I would also like to extend my sincere thanks to Justin Whatley, Research Scientist, EXFO Inc., for his cooperation and assistance in data collection and preprocessing. His passionate participation and curiosity provided me with valuable insights and a contagious enthusiasm for this subject.

Contribution of Authors

This thesis is under submission as Bordeau Aubert, Whatley, Nadeau, Glatard, and Jaumard (2023). This research project had several coauthors. I elaborated the methodology, conducted the experiments, and drafted the article and thesis. Sylvain Nadeau, Tristan Glatard and Brigitte Jaumard guided me with their expertise as supervisors. They were invaluable in delimiting the study's direction and ensuring the research adhered to academic standards. They also reviewed the article and thesis and provided feedback to enhanced their clarity and quality. Justin Whatley provided the real-world network data and contributed in the technical field by explaining the tools and time series analysis used by EXFO.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Communication networks generates massive amounts of data capturing various network activities and behaviors. Network Key Performance Indicators (KPI) analysis allowed network optimization, anomaly detection, and predictive maintenance to ensure seamless network operations. Time series analysis is used in traffic analysis and performance monitoring. Anomaly classification in network KPI is an important research area aimed at categorizing unusual events within network data. Detection and classification of anomalies play a crucial role in evaluating and ensuring the stability, and reliability of network systems. As such, understanding anomalies on KPI allows network operators to maintain optimal network performance and prevent faults.

Despite the importance of both anomaly classification and anomaly detection in time series, a notable disparity of research focus between these two areas was observed in recent years. While anomaly detection, aimed at identifying unusual patterns in network data, has gathered considerable attention and resulted in several algorithms and techniques, the field of anomaly classification, which involves categorizing anomalies into specific classes, has not received the same attention in time series. This discrepancy is disturbing because anomaly classification can significantly enhance network efficiency, helping network administrators to distinguish between benign and impactful anomalies. To bridge this gap, more research efforts are needed in anomaly classification to develop robust and effective methods that complement existing anomaly detection approaches, fostering a more comprehensive network analysis strategy.

This study focuses on simulating anomalies on network KPI and evaluating anomaly classification models using both simulated and real-world noisy time series datasets. The framework assumes separate entities for the classifiers and the detector within the anomaly classification process. This separation allows for a modular approach where the detection of anomalies and the classification of those anomalies are treated as distinct tasks. The primary objectives of this thesis are to develop a framework containing a network KPI time series simulator, detection and classification models, and to evaluate their performance on both simulated and real datasets. By generating simulated time series with anomalies, we aim to mimic real-world network KPI, providing a valuable tool for anomalies detection and classification. The framework also aims to enhance network monitoring through the detection of anomalies, while the classification models seek to accurately categorize these anomalies. We evaluate these models on both simulated and actual network KPI datasets to assess their performance and generalization capabilities.

The thesis follows the structure of a conference paper starting from Chapter 3, providing in-depth analyses and research findings presented in that article. This article was submitted as an arXiv pre-print and to the IEEE Access journal. Chapter 2 provides a comprehensive review of the literature concerning network KPI anomaly definition, models for anomaly detection and classification, and evaluation methodologies. In Chapter 3, we detail our anomaly classification framework, covering simulation, detection, and classification models. Finally, Chapter 4 presents the experimental setup, encompassing dataset utilization and model performance evaluation across both simulated and real-world time series datasets, followed by concluding remarks in Chapter 5.

Chapter 2

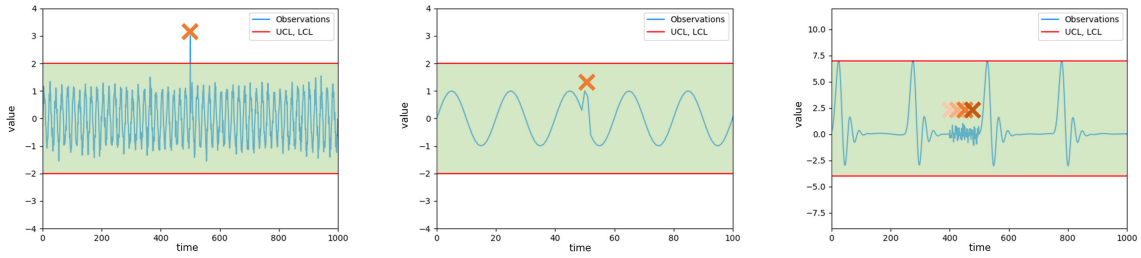
Background

Anomalies pose significant challenges to the stability and performance of network systems. In recent years, extensive research has been conducted to detect these anomalies. This background chapter aims to provide an overview of the key concepts, methodologies, and advancements in the field of anomaly definition, simulation, detection, and classification.

2.1 Anomaly definition

Defining network KPI anomalies is the first step in the creation of a simulator and classification model. Anomalies encompasses a wide range of definitions and interpretations, reflecting the diverse nature of anomalies that can occur in network systems.

Choi et al. (2021) identified 3 major anomaly types in time series: point, contextual, and collective anomalies as illustrated in Figure 2.1. Point anomalies refer to a data point or sequence that exhibits a sudden and significant deviation from the normal range. These anomalies are typically caused by sensor errors or abnormal system operations and are detected by comparing values against predefined upper and lower control limits. Contextual anomalies present a challenge for detection as they do not deviate from the normal range based on predefined limits. This type of anomaly is characterized by a group of points that does not contains extreme values, this anomalies modify the normal shape of the signal. Collective anomalies represent a set of data points that gradually display



(a) Point anomaly (b) Contextual anomaly (c) Collective anomaly

Figure 2.1: Anomaly types extracted from Choi et al. (2021)

a different pattern from normal data over time. While individual values within this type may not appear anomalous, their collective behavior raises suspicion. Detecting collective anomalies requires examining long-term contexts to identify deviations from the expected pattern.

Foorthuis (2018, 2021) proposed the following five fundamental data-oriented dimensions for describing types and subtypes of anomalies: data type, cardinality of relationship, anomaly level, data structure, and data distribution as illustrated in Figure 2.2. These dimensions contribute to the classification and characterization of anomalies in network data. The data type dimension differentiates between quantitative, qualitative, and mixed attributes. The cardinality of relationship dimension distinguishes between univariate and multivariate relationships among attributes. The anomaly level dimension classifies anomalies as either atomic (individual low-level cases) or aggregate (groups or collective structures). The data structure dimension considers different structural formats, such as graphs and time series, which host specific anomaly subtypes. Lastly, the data distribution dimension focuses on the collection and pattern of attribute values in the data space, providing additional descriptive and delineating capabilities for anomaly classification. We focused our research on quantitative and univariate time series with atomic and aggregate anomalies. The anomalies we focused our attention in this thesis are the local additive anomaly, the temporary change anomaly, the level shift anomaly and the variation change anomaly.

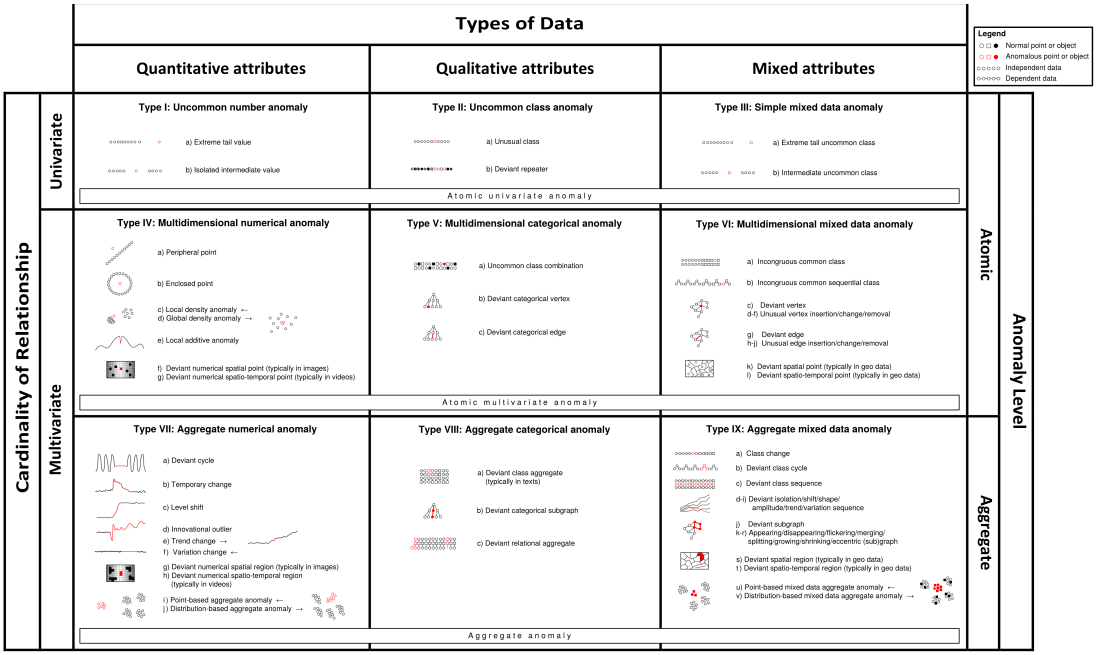


Fig. 3: The typology including all types and subtypes. Each anomaly subtype is represented by an icon that depicts the essence of the deviation. An icon that includes lines represents a set with dependent data. (Zoom in on a digital screen to see details.)

Figure 2.2: Table extracted from Foorthuis (2021)

2.2 Anomaly detection

Anomaly detection techniques aim to identify abnormal patterns or events within time series data. Traditional approaches include statistical methods, rule-based systems, and expert systems, which rely on predefined thresholds or rules to flag deviations. Forecasting is one of the most used techniques for anomaly detection in network KPI time series, as it enables real-time analysis. A popular forecasting approach is based on the Autoregressive Integrated Moving Average (Box & Pierce, 1970) (ARIMA) model, which combines autoregressive (AR), integrative (I), and moving average (MA) components to capture the underlying patterns and characteristics of a time series. When evaluating and forecasting stationary time series data, ARIMA models are useful because they produce predictions for the future by considering both historical data and prediction errors.

Deep learning techniques for time series forecasting, including Recurrent Neural Networks (RNN) (Elman, 1990; Madan & Mangipudi, 2018) and Convolutional Neural Networks (CNN) (Borovykh, Bohte, & Oosterlee, 2017; LeCun, Bottou, Bengio, & Haffner, 1998), have also shown promise in capturing temporal dependencies and spatial patterns for improved anomaly detection,

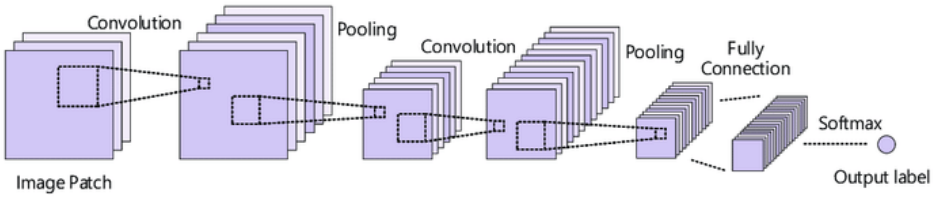


Figure 2.3: CNN structure extracted from Mei et al. (2019)

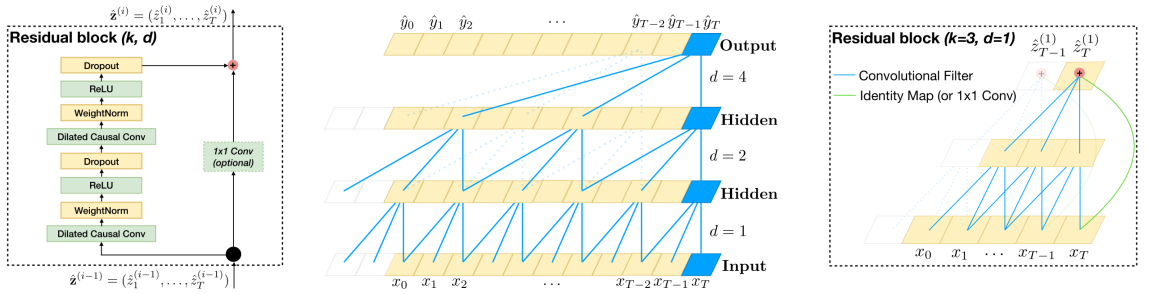
as explained by Choi et al. (2021).

RNN-based models are widely used in speech or voice recognition, natural language processing, and time series prediction. This type of neural network is recurrent in nature as it uses the output of a previous layer to feed the current input. They are robust to the noise and are able to learn long-term dependencies and temporal patterns. However, RNN models are subject to the gradient vanishing problem leading to slow training time and issue in learning long-term dependencies. To address this issue, Long-Short Term Memory (LSTM) was proposed by Malhotra, Vig, Shroff, Agarwal, et al. (2015). This study show that LSTM networks can effectively learn temporal patterns and detect anomalies without prior knowledge of pattern duration. The LSTM approach demonstrates promising results on real-world datasets, outperforming or matching RNN and indicating the robustness of LSTM-based models in capturing both short-term and long-term dependencies in normal time series behavior.

CNN models are mainly used in image processing and recognition tasks. CNN architectures are composed of three types of layers: the convolutional layers, the pooling layers and the fully connected layer. The convolutional layers use filters to identify the pattern of the image or time series. Each filter check the image for a single feature and the Rectified Linear Unit (ReLU) function is used to transform the features after each convolution. The pooling layers limits the complexity of the networks by reducing the dimensionality of the input. The fully connected layer is the classification layer, it uses the features from previous filters and layers to classify the image or time series with a softmax function. CNN are also robust to noise and minnimize the computation. However, CNN needs large dataset and takes a long time to train. See illustration in Figure 2.3 by Mei et al. (2019).

More recently, the need to combine CNN and RNN approaches in time series detection was

seen as essential (Xue, Triguero, Figueredo, & Landa-Silva, 2019), as it allows for the simultaneous extraction of spatial features and temporal dependencies. This combination enables more comprehensive and accurate analysis of time-varying patterns in the data. Lea, Vidal, Reiter, and Hager (2016) proposed the Temporal Convolutional Network (TCN) as a unified approach between RNN and CNN architectures. The TCN demonstrates comparable or better performance than other models on various datasets, and it offers the advantage of faster training. Bai et al. (2018) highlights the superiority of TCNs over generic recurrent architectures, such as LSTM and Gated Recurrent Units (GRU) (Cho et al., 2014), in various sequence modeling tasks. The TCN model, incorporating dilations, residual connections, and causal convolutions, consistently outperforms recurrent architectures. A baseline TCN contains a single residual block. This block is defined by two layers of dilated causal convolution. Each layer is composed of a weight normalization for the convolutional filters, a ReLU, and a spacial dropout. The dilated causal convolution allows for larger receptive fields and facilitates the ability to access more distant historical information within the time series compared to a basic causal convolution. To ensure the same size between the input and the output, the residual block also use an additional 1x1 convolution. See illustration in Figure 2.4a. Figure 2.4c shows an example of a dilated causal convolution with a dilation factors $d = 1, 2, 4$ and a filter size $k = 3$. Figure 2.4c illustrates a residual block with a filter size $k = 3$ and a dilation factor $d = 1$. The study also challenges the notion that recurrent networks have an inherent advantage in preserving long-range dependencies, as TCN demonstrate comparable or even longer memory capabilities. Overall, the results suggest that convolutional networks, with their simplicity and clarity, should be considered a strong foundation and a powerful toolkit for sequence modeling. We used the TCN model for anomaly detection and classification purpose in this thesis.



(a) Residual block (b) Dilated causal convolution (c) Example of residual connection

Figure 2.4: Architectural elements in a TCN extracted from Bai et al. (2018)

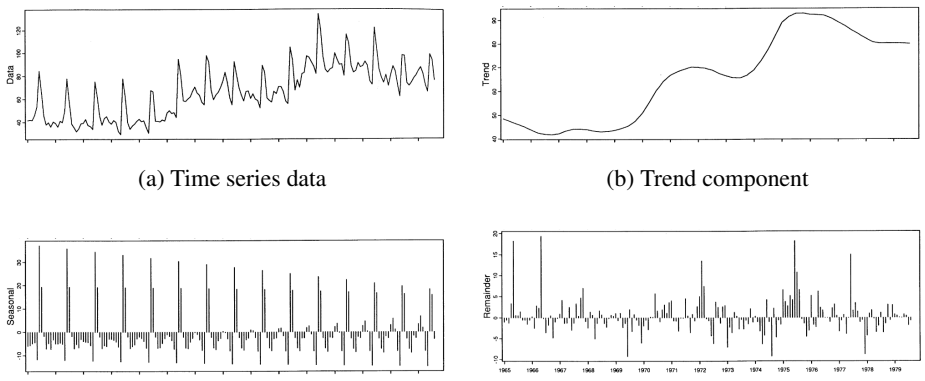
2.3 Anomaly classification

Multiple preparation over the time series were proposed over the years, one preparation often used is the decomposition of the time series into three or more components: the seasonality, the trend and the residuals. The classical decomposition is the moving average method. This decomposition can either be additive or multiplicative:

$$X = T_i + U_i + R_i \qquad X = T_i \times U_i \times R_i,$$

where X is the time series, T_i is the trend component, U_i is the seasonal component, and R_i is the residual (or remainder). To evaluate this components, the moving average decomposition has to (1) evaluate the trend component of the time series using moving average, (2) detrend the initial time series, (3) average the values of the detrended time series to find the seasonal component, (4) extract the remainder or residual by subtracting the trend and seasonal component to the initial time series.

The Seasonal and Trend decomposition using Loess (Cleveland et al., 1990) is often used in time series decomposition as it handle any types of seasonality (weekly, monthly, etc.). It offers various benefits, including user-controlled trend smoothing, an adaptable seasonal component, and resilience against outliers. This decomposition handles both additive and multiplicative decomposition See illustration in Figure 2.5.



(a) Time series data

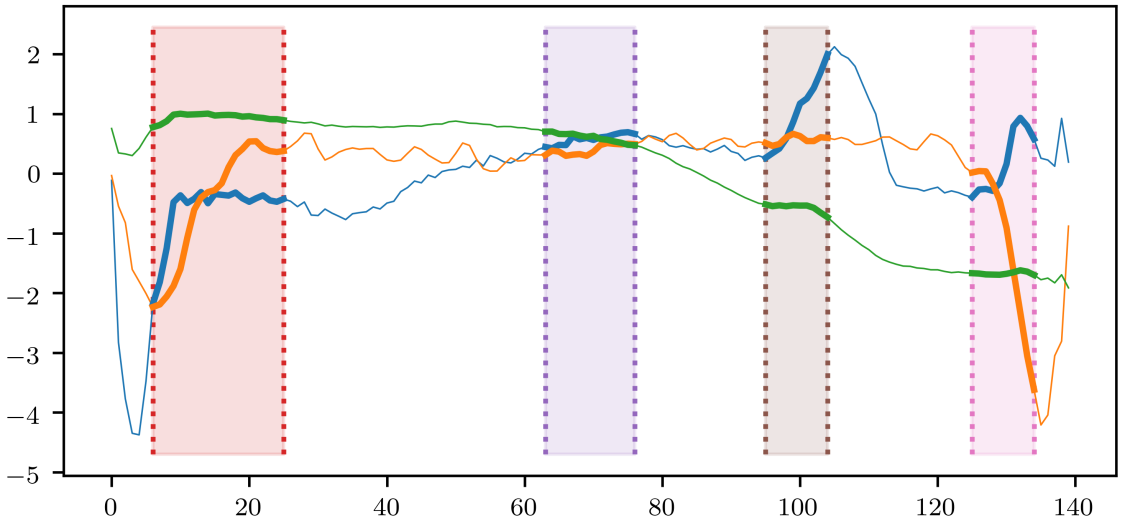
(b) Trend component

(c) Seasonal component

(d) Remainder component

Figure 2.5: Illustration of STL decomposition extracted from Cleveland et al. (1990)

Anomaly classification on KPI focuses on categorizing detected anomalies into meaningful classes. Classification models utilize features extracted from network KPI data to differentiate between different types of anomalies, such as intrusion attempts, Denial of Service (DoS) attacks, or network performance issues. Faouzi (2022) reviewed the algorithm and implementation of Time series classification (TSC). The study described various approaches such as the Nearest neighbors, a widely adopted and well-established method in time series classification is the Nearest Neighbor (NN) (Bagnall & Lines, 2014) classifier combined with the Dynamic Time Warping (DTW) distance. Extensive comparisons conducted by Bagnall, Lines, Bostrom, Large, and Keogh (2017) confirmed that DTW outperforms other distance measures. Another approach was with tree-based algorithm, such as the Time series Forest (TSF) (Deng, Runger, Tuv, & Vladimir, 2013) as illustrated in Figure 2.6.



(a) Interval visualization

	Interval 1			Interval 2			Interval 3			Interval 4		
	Mean	SD	Slope	Mean	SD	Slope	Mean	SD	Slope	Mean	SD	Slope
Time series 1	-0.61	0.504	0.052	0.58	0.086	0.02	1.004	0.572	0.197	0.189	0.504	0.156
Time series 2	-0.467	0.977	0.158	0.403	0.084	0.017	0.568	0.061	0.011	-1.204	1.305	-0.431
Time series 3	0.944	0.061	0.002	0.604	0.075	-0.018	-0.57	0.065	-0.017	-1.671	0.024	0.003

(b) Table with the features extracted from the interval

Figure 2.6: Illustration of TSF extracted from Faouzi (2022)

Cabello et al. (2020) introduced the Supervised Time series Forest (STSF) as an evolution of the TSF, an efficient algorithm for interval-based time series classification on high-dimensional datasets. STSF is a decision tree ensemble that utilizes class-balanced bagging to sample the training set. As shown in Figure 2.7, STSF generates intervals for three representations (time series, frequency domain, and derivative representation of the time series) and extracts seven features (mean, median, standard deviation, slope aggregation functions, inter-quartile range, minimum, and maximum) from these representations. The frequency domain time series is obtained after derivation from the discrete Fourier transform. The advantage of this representation is that it improves in the indirect detection of phase independent discriminating intervals. The Derivative representation use a first order difference to achieve better performance during the classification process. STSF leverages multiple time series representations, a supervised search strategy, and a feature ranking metric to reduce computational overhead while identifying highly discriminatory interval features

for interpretable classification outcomes. The paper stated STSF achieved comparable accuracy to state-of-the-art methods in time series classification but with significantly faster processing times, enabling classification of large datasets with long series.

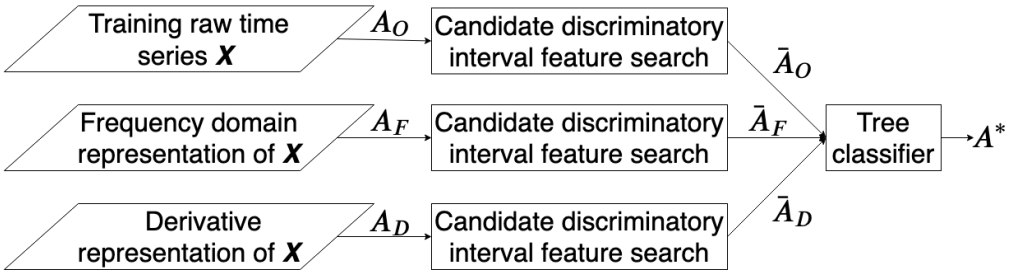


Figure 2.7: STSF overview extracted from Cabello et al. (2020)

Anomaly classification models also witnessed a transition from traditional rule-based and statistical approaches to more advanced techniques such as machine learning, deep learning, and ensemble methods, enabling improved accuracy and generalization capabilities in detecting and categorizing anomalies. Zhao, Lu, Chen, Liu, and Wu (2017) suggested that a CNN-based method for time series classification outperforms competing baseline methods in terms of classification accuracy and noise tolerance. By automatically discovering and extracting the internal structure of input time series, the CNN generates deep features that improve classification performance. The study also discusses the significance of three CNN parameters: convolutional filter size, pooling method, and the number of convolution filters. However, limitations are acknowledged, including the time-consuming nature of CNN training due to parameter experimentation and the fixed length requirement for time series during training and testing. Ongoing research aims to address these limitations by designing optimal parameters and exploring a new network architecture that combines CNN with RNN. Ismail Fawaz et al. (2020) introduced InceptionTime, a CNN-based classifier achieving state-of-the-art performance UCR Chen et al. (2015) archive datasets. This study also discusses the importance of the parameters such as the depth (a deeper network gives better performance) and the number of filters that should also be limited as it deteriorates the classifier performance. Even though numerous studies on time series classification have been carried out, more research and analysis are still needed in the specific area of network KPI anomaly classification. Studies concentrating solely on network KPI anomaly classification in time series data are

noticeably scarce, with little study devoted to this particular field. Although anomaly detection is extremely important, research into and development of classification models specifically suited for network KPI time series data is still in its early stages.

Chapter 3

Anomaly Classification Framework

This section describes the simulation method to generate anomalies, as well as the detection and classification models used in the framework. Our framework separates anomaly detection from anomaly classification as shown in Figure 3.1.

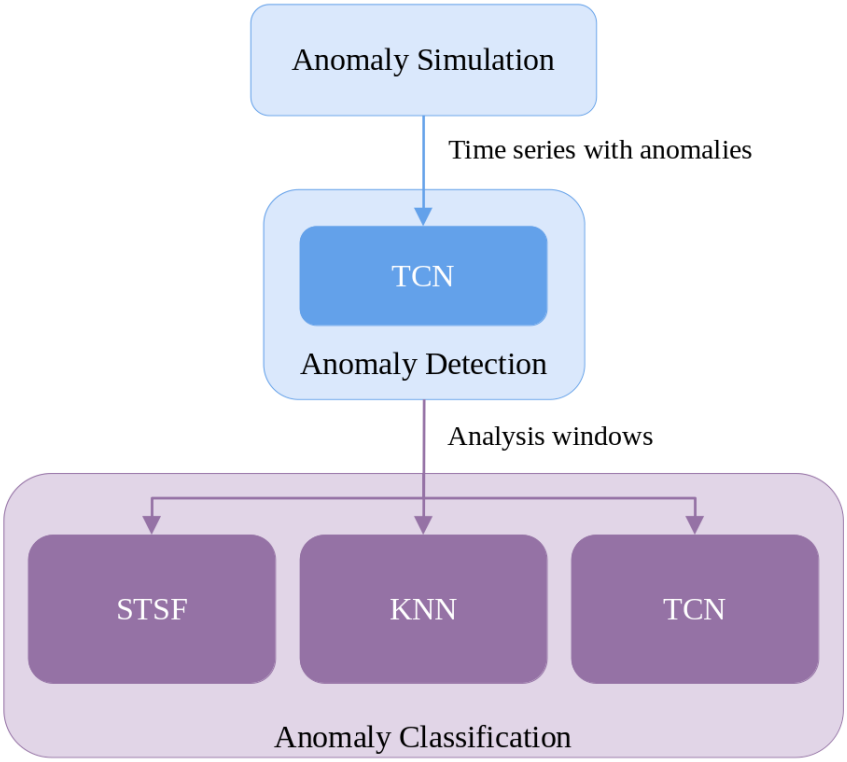


Figure 3.1: Anomaly classification framework

n	number of anomalies to inject
X	initial time series
$\tilde{X}$	time series with anomalies
$\bar{X}$	simulated noise time series
$\hat{X}$	final time series with anomalies and noise
$\check{X}$	predicted time series
A	amplitude of the sine signals
T	seasonality period of the sine signals in minutes
t_s	sampling period
μ	mean of the sine signals
$\bar{\mu}$	mean of the normal distribution for the noise generation
$\bar{\sigma}$	standard deviation of the normal distribution for the noise generation
σ	noise level of the time series
A_d	daily amplitude of the time series
λ	length of the anomalies
α	strength of the anomaly
i_w	first index of an anomaly window
δ	confidence interval to determine a point as anomalous
m	margin period of the analysis windows
T_i, U_i, R_i	time series features (seasonal, trend and residual)

Table 3.1: Notations

3.1 Anomaly Simulation

Anomaly simulation aims to generate network Key Performance Indicators (KPI) time series. Network KPIs are used by companies to track network performance from metrics such as packet loss or latency. These indicators gives an overview of the network usage and health. In this paper, we focus on latency, which is typically modeled with 3 seasonality components (daily, weekly, and monthly seasonality), a trend, and a noise component (Fig. 3.2). The latency is commonly measured as follows: (1) a packet is sent from point A to B in the network, (2) the packet is sent back from B to A, (3) steps (1) and (2) are repeated for a given number of packets, (4) the latency is obtained by averaging the packet round-trip time between A and B. We denote the data sets respectively as SIM for the time series resulting from anomaly simulation and REAL for the real time series data sets.

3.1.1 Signal Generation

We define an *anomaly window* as a sub-sequence of the original time series containing exactly one anomaly. We also define n the number of anomalies to inject in the SIM time series. We

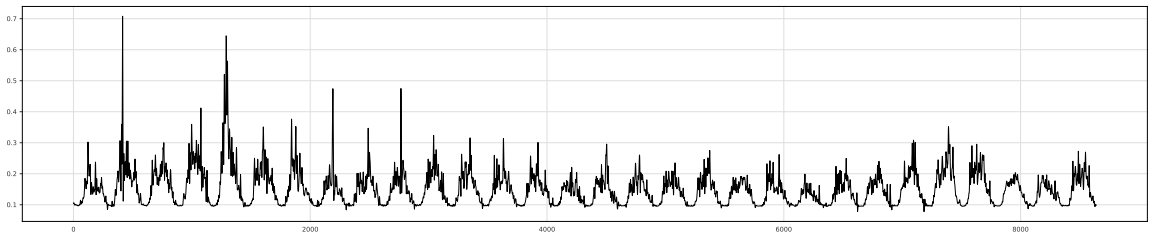


Figure 3.2: Example of REAL time series after processing

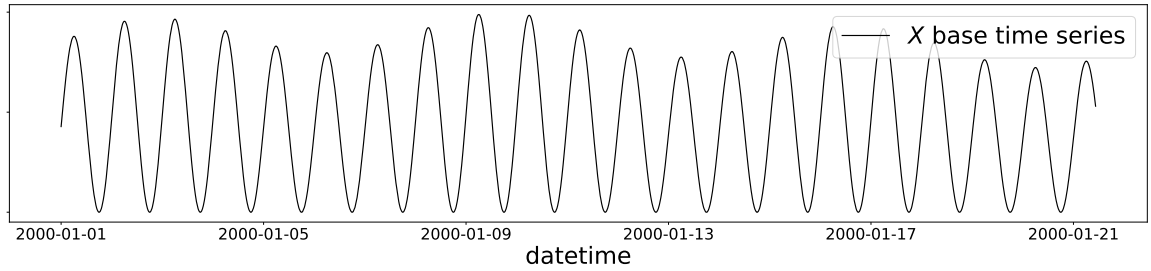


Figure 3.3: X base time series.

generate n anomaly windows, as follows: (1) we randomly select the anomaly class to generate based on anomaly proportions set as a parameters; (2) we set the anomaly window size to $2Y$ points where Y is a random variable following the uniform distribution $\mathcal{U}(l_{min}, l_{max})$. We empirically set the limits l_{min}, l_{max} depending on the anomaly type. The size of the SIM time series corresponds to the sum of each anomaly window size. We generate the SIM base time series using 3 seasonality components modeled with 3 sine signals with daily, weekly and monthly periodicity:

$$X(t) = \prod_{s=0}^{2} \left(A_s \times \sin\left(\frac{2\pi}{T_s} t\right) + \mu_s \right),$$

where A_s is the amplitude: $A = (A_0, A_1, A_2) = (0.5, 0.1, 0.05)$, A represent in reality the average delay in μs; T_s is the seasonality period expressed in minutes: $T = (T_0, T_1, T_2) = (1440, 10080, 40320)$; μ_s is the mean of the sine signal: $\mu = (\mu_0, \mu_1, \mu_2) = (0.5, 0.9, 0.95)$; and t is expressed in minutes. The amplitudes and means add up to 1, to keep the base time series between 0 and $1\mu s$ which corresponds to typical latency values. The different amplitudes are set according to observations on the REAL data sets. The shape of this base time series is consistent with common observations of network traffic that usually include these pseudo-periods without noise. The time series X is then uniformly sampled with the period t_s (Fig. 3.3).

3.1.2 Anomaly Injection

We define each anomaly type with 2 parameters:

(1) The anomaly length λ, defined as the number of points modified during injection in the time series. To select the length of each anomaly, we randomly select a starting index i_a in $[i_w, i_w + 2Y - e]$ where i_w is the index of the first point in the anomaly window and e corresponds to the minimum margin $e = 5$ that ensures a minimal size for each class of anomalies. λ is then randomly selected in $[i_a, i_w + 2Y - e]$.

(2) The strength α, which depends on the daily amplitude A_d:

$$A_d = \max_{i \in d} X(i) - \min_{i \in d} X(i),$$

where d is the daily range such that $i_a \in d$. d corresponds to a daily period of length $\frac{1440}{t_s}$ minutes. α is then defined as:

$$\alpha = A_d \times Z,$$

where Z is a random variable following the uniform distribution $\mathcal{U}(0.5, 0.7)$.

We denote by $\tilde{X}$ the time series containing a simulated anomaly. The 4 simulated types of anomalies are the following:

Single point (peak or dip) This anomaly type has a length λ equal to 1 point and Y follows the uniform distribution $\mathcal{U}(120, 480)$. The anomaly window is defined for i in $[i_w, i_w + 2Y]$ as:

$$\tilde{X}(i) = X(i) \pm \begin{cases} \alpha & \text{if } i = i_a \\ 0 & \text{otherwise,} \end{cases}$$

In this formula, peaks are generated by the addition and dips by the subtraction. See illustration in Figure 3.4a.

Temporary change (growth or decrease) The length of this anomaly type varies in $[3, i_w + 2Y]$ and Y follows the uniform distribution $\mathcal{U}(240, 960)$. The anomaly is described as a growth or

decrease that progressively reverts to the original signal:

$$\tilde{X}(i) = X(i) \pm \begin{cases} (i - i_a)\dfrac{\alpha_1}{i_b - i_a} & \text{if } i \in [i_a, i_b] \\ (i - i_b)\dfrac{\alpha_2 - \alpha_1}{i_c - i_b} + \alpha_1 & \text{if } i \in [i_b, i_c] \\ (i - i_a - \lambda)\dfrac{\alpha_2}{i_c - i_a - \lambda} & \text{if } i \in [i_c, i_a + \lambda] \\ 0 & \text{otherwise,} \end{cases}$$

where i_b is randomly selected in $[i_a,\, i_a + \lambda/2]$, i_c is randomly selected in $[i_b,\, i_a + \lambda]$, and α_1, α_2 are either set to α and between $[0.4, \alpha]$ or vice versa. See illustration in Figure 3.4b.

Level shift (growth or decrease) The length of this anomaly type varies in $[3, i_w + 2Y]$ and Y follows the uniform distribution $\mathcal{U}(1440, 2160)$. This type of anomaly is a variant of the previous one where $\alpha_1 = \alpha_2$. It is defined by a change of the time series trend:

$$\tilde{X}(i) = X(i) \pm \begin{cases} (i - i_a)\dfrac{\alpha}{i_b - i_a} & \text{if } i \in [i_a, i_b] \\ \alpha & \text{if } i \in [i_b, i_c] \\ (i - i_a - \lambda)\dfrac{\alpha}{i_c - i_a - \lambda} & \text{if } i \in [i_c, i_a + \lambda] \\ 0 & \text{otherwise,} \end{cases}$$

where i_b is randomly selected in $[i_a,\, i_a + \lambda/2]$, and i_c is randomly selected in $[i_b,\, i_a + \lambda]$. See illustration in Figure 3.4c.

Variation change (growth or decrease) The length of this anomaly type varies in $[3, i_w + 2Y]$ and Y follows the uniform distribution $\mathcal{U}(1440, 2160)$. This type of anomaly amplifies or reduces the amplitude of a time series while preserving its period:

$$\tilde{X}(i) = X(i) \pm \begin{cases} X(i)\alpha & \text{if } i \in [i_a, i_a + \lambda] \\ 0 & \text{otherwise,} \end{cases}$$

See illustration in Figure 3.4d, where the signals only include daily seasonality (weekly and monthly seasonalities are omitted) to simplify presentation. The simulator supports 4 anomaly types where each type can be separated into 2 sub-classes (growth or decrease).

3.1.3 Data Preparation and Noise Injection

Scaling We apply the MinMaxScaler[1] function from the scikit-learn library to scale the time series to $[0.02, 1]$. The minimum is fixed to 0.02, which is the range observed in the real dataset. Indeed, real latency values can never reach exactly 0. This scaler assure the time series are between the same range for both the detection and classification for the SIM and REAL data sets.

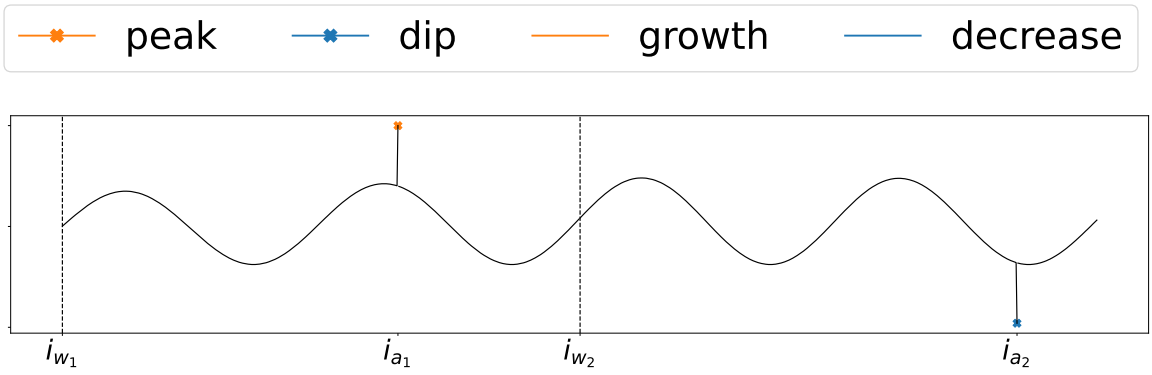
Noise Injection We create a noise time series $\bar{X}$ with equal length to $\tilde{X}$. This noise is defined by a Gaussian multiplicative white noise:

$$\bar{X}(i) = X(i) \times \begin{cases} \min(W, 4\bar{\sigma}) & \text{if } W \geq 0 \\ \max(W, -4\bar{\sigma}) & otherwise, \end{cases}$$

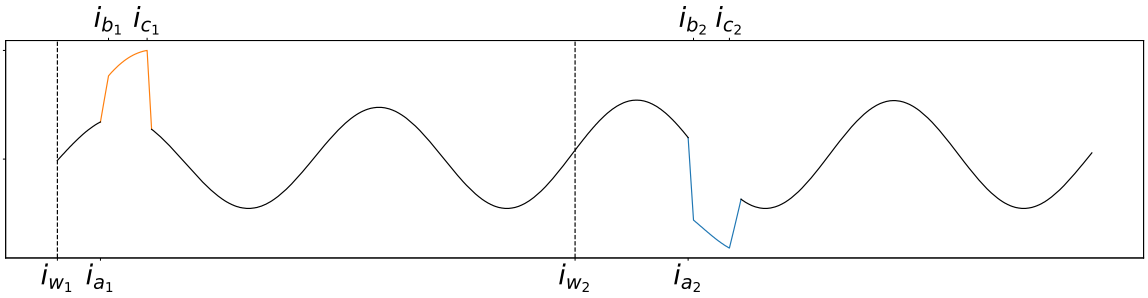
where W is a random variable following the normal distribution $\mathcal{N}(\bar{\mu}, \bar{\sigma})$, and $\bar{\sigma}$ is the standard deviation of the simulated Gaussian white noise time series, see illustration in Figure 3.5. We remove the extreme values to evaluate the anomalies injected by the simulator and not those resulting from the noise. We separate the noise level σ of the time series from the standard deviation $\bar{\sigma}$ of the noised time series. Indeed, as we multiplied the Gaussian white noise time series by $X(i)$, the noise level σ extracted from the final time series isn't equal to the noise level $\bar{\sigma}$ we used during the noise generation. Indeed, due to the seasonal shape of the initial time series, we obtained $\bar{\sigma} = c \times \sigma$ with the constant $c = 2.31$.

The Single point and Temporary change anomalies aren't affected by the noise as they are part of the same frequencies. Indeed, the noise is already contained in these types of anomalies.

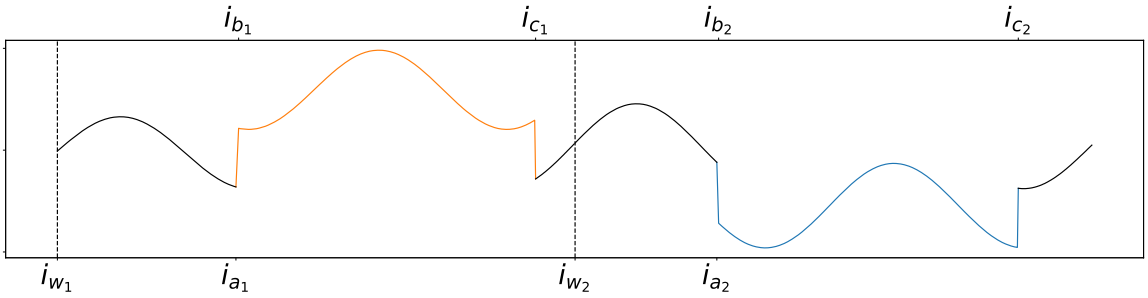
[1] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html



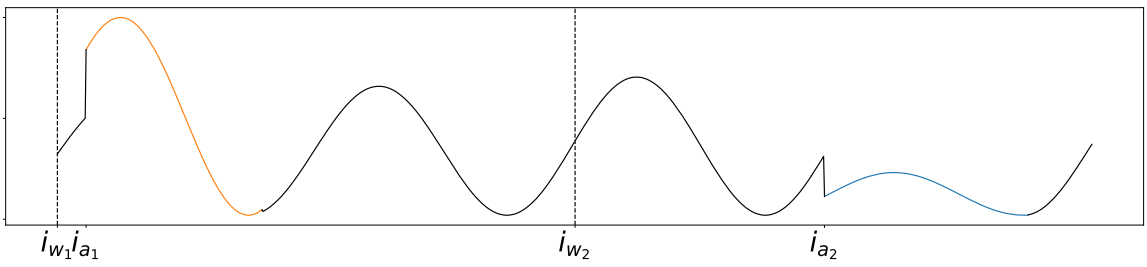
(a) Single point



(b) Temporary change



(c) Level shift



(d) Variation change

Figure 3.4: Types of anomalies.

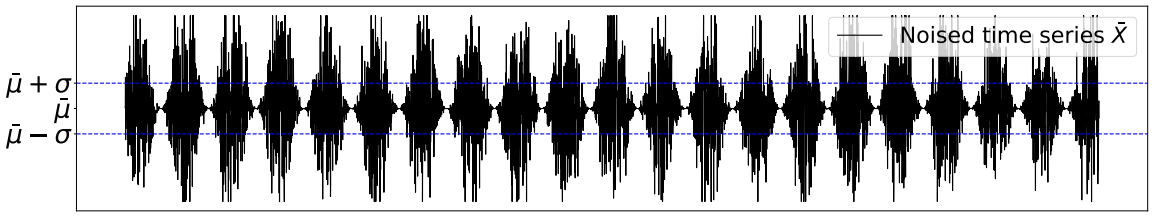


Figure 3.5: Noised time series $\bar{X}$

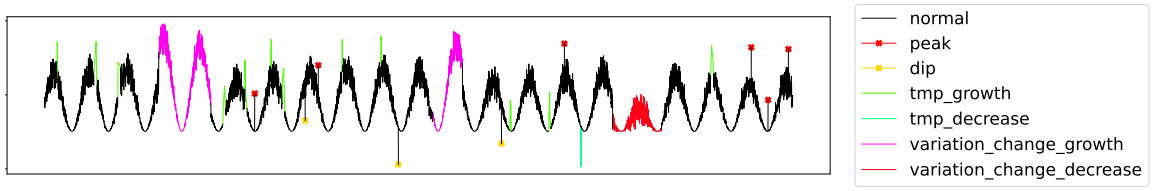


Figure 3.6: Final simulated time series (SIM $\hat{X}$)

We denote $\hat{X}$ the time series resulting from the scaling and noise injection:

$$\hat{X}(i) = \tilde{X}(i) + \begin{cases} 0 & \text{if } \tilde{X}(i) \in \text{Single point} \\ 0 & \text{if } \tilde{X}(i) \in \text{Temporary change} \\ \bar{X}(i) & \text{otherwise,} \end{cases}$$

The resulting time series $\hat{X}$ is used in the SIM data sets, as illustrated in Figure 3.6. $\hat{X}$ also corresponds to the processed REAL data sets. The parameters used to generate the anomalies are the noise level, the proportion of each type of anomalies, and the sampling period t_s. The sampling period is an important parameter of the simulation. For instance, a Single point anomaly simulated for a given sampling period may be interpreted as a Temporary change for another sampling period.

Parameter	Values tested
Kernel size	2, 3, 4, 5, 6, 7
Number of filters	4, 5, 6, 7, 8
Dilation base	3, 4, 5
Input chunk length	$(1440/t_s) \times 8$
Output chunk length	$(1440/t_s) \times 7$

Table 3.2: Parameter combinations tested for the TCN time series parameter.

3.2 Anomaly Detection

To evaluate the anomaly *classification* model, we build an anomaly *detection* model using a Temporal Convolutional Network (TCN Bai et al. (2018)), a common approach to time series prediction (Torres, Hadjout, Sebaa, Martínez-Álvarez, & Troncoso, 2021). We used the TCN implementation of the darts library[2], varying the parameters according to Table 3.2.

Detection model The TCN detection model is trained for 10 epochs with the Adam optimizer. We limit the number of residual block to 2 to reduce the training time. This model is composed of ReLU activation functions with a β-Negative log likelihood loss (Seitzer, Tavakoli, Antic, & Martius, 2022) from the Gaussian likelihood function[3]. The Gaussian likelihood function provides boundaries for the predicted time series with a 95% confidence interval δ. The detection model gives a weekly prediction from the previous 8 days.

To train and test the detection model, we split $\hat{X}$ using the function TimeSeriesSplit from the scikit-learn library[4]. This function creates multiple training and test sets time series as illustrated in Figure 3.7. We split $\hat{X}$ with ten folds and fix the training/test size to $70\% - 30\%$.

For each fold of the TimeSeriesSplit function, we apply the fitted TCN model to the test set resulting in the predicted time series $\check{X}$ and the confidence interval δ. We label a point of $\hat{X}$ as anomalous if and only if:

[2] https://unit8co.github.io/darts/generated_api/darts.models.forecasting.tcn_model.html

[3] https://unit8co.github.io/darts/generated_api/darts.utils.likelihood_models.html#darts.utils.likelihood_models.GaussianLikelihood

[4] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html

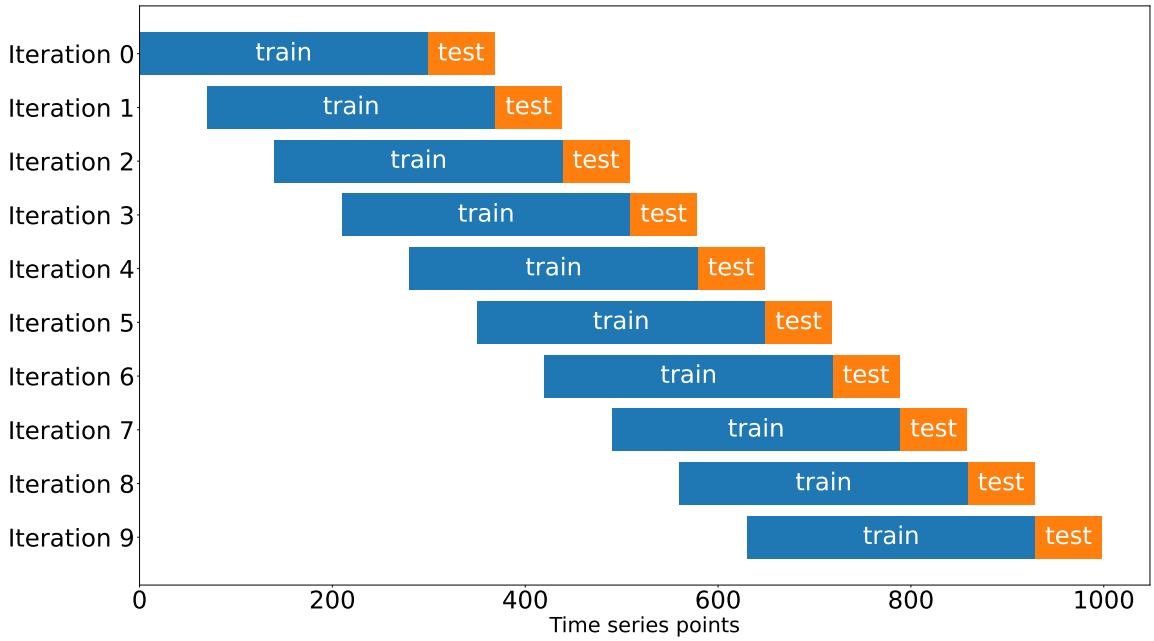


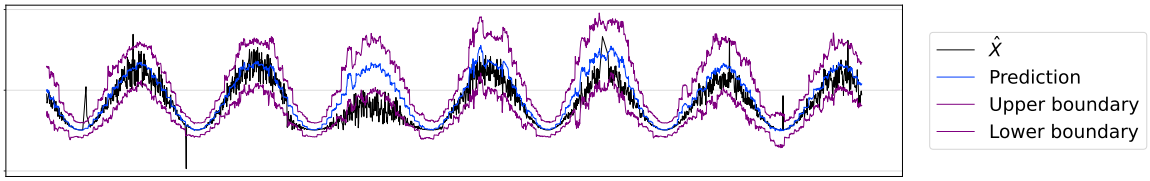
Figure 3.7: Data partitioning in training and test sets

$$\left|\hat{X} - \check{X}\right| > \delta,$$

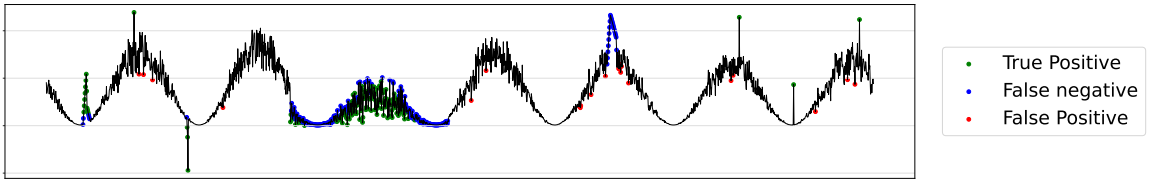
See illustration in Figure 3.8.

Analysis windows We denote aSIM and aREAL the simulated and real analysis windows resulting from the anomaly detection model. An analysis window is a sub-sequence time series of $\hat{X}$ of fixed size $2m$, where m is the margin period. The default sampling of the time series is set to 1 point/minute. The initial size of an analysis windows is set to 240 points (4 hours of data centered on the anomaly) with m corresponding to 2 hours of data (120 points). Taking a sampling period $t_s = 5$ minutes (which creates a time series X with a point each 5 minutes), the margin size of the analysis windows are set to $m = 24$ points. This margin size is used to center the anomaly in the analysis window. We create the analysis windows from the test set during the cross-validation. We denote S the list containing the point states with the states "normal" or "anomaly". We create the analysis windows as illustrated in Algorithm 1.

It should be noted that (1) a given analysis window may contain multiple anomalous points, (2) the anomalous points found in a given analysis window may be related to different anomalies,



(a) Prediction and Boundaries



(b) Points status decision

Figure 3.8: TCN anomaly detection

Algorithm 1: Analysis windows creation

Input : $\hat{X}$ the time series with anomalies
S the list containing the state of each point (”normal” or ”anomalous”)
m the analysis windows margin

Output: a the analysis windows list aSIM or aREAL

$i \leftarrow 0$
$a \leftarrow$ empty list
while *not at the end of* S **do**
  if $S(i)$ *is ”anomaly”* **then**
    $n_c \leftarrow$ number of consecutive points detected as ”anomaly”
    new$_a \leftarrow$ sub-sequence time series $\hat{X}_{[i-m,i+m]}$
    $a \leftarrow a +$ new$_a$
    $i \leftarrow i + n_c$
  else
    $i \leftarrow i + 1$
  end
end

(3) analysis windows may overlap (meaning we could have the same anomaly present in different analysis windows), (4) when simulated data is used, analysis windows are defined independently from the anomaly windows produced by the simulator. We evaluate our anomaly detection system by computing its F1 score from the following measures:

- True positives (TP): the number of true anomalies that are included in at least one analysis window.
- False positives (FP): the number of analysis windows that do not contain any true anomaly.
- False negatives (FN): the number of true anomalies that are not included in any analysis window.

This definition of the F1 score is sensitive to the size $2m$ of an analysis window. Indeed, increasing $2m$ mechanically increases the F1 score by reducing the number of false positives and false negatives, and increasing the number of true positives.

3.3 Anomaly Classification

Time series decomposition During the training of the detector (Section 3.2), we also apply the additive Moving average time series decomposition[5] from the statsmodel library:

$$\hat{X} = T_i + U_i + R_i,$$

where $\hat{X}$ is the time series containing anomalies, T_i is the trend component, U_i is the seasonal component, and R_i is the residual. As the seasonality amplitude could vary each week in real uses, we decided to extract the seasonality from the last 2 weeks (as we need a minimum of 2 occurrences to extract the seasonality) of the train time series. We also extracted the trend from the last week of the train time series. This decomposition is done to evaluate the analysis windows labels based on the last observations of the train time series (as the detection model gives a week prediction based on the last 8 days). The seasonality and the trend extracted are subtracted to

[5] https://www.statsmodels.org/dev/generated/statsmodels.tsa.seasonal.seasonal_decompose.html

the test set. This decomposition is applied after the prediction and before the analysis windows creation. The classifiers are then trained on R_i. We use the Moving average decomposition because it gives a more stable seasonality compared to the STL (Seasonal and Trend decomposition using Loess (Cleveland et al., 1990)) and more versatile than traditional decomposition methods such as X11 and SEATS (Seasonal Extraction in ARIMA Time Series) (Dagum & Bianconcini, 2016; Hyndman & Athanasopoulos, 2018).

Classification models We tested a total of three classification models, including two classical machine-learning models, namely k-Nearest Neighbors (kNN) and Supervised Time Series Forest (STSF Cabello et al. (2020)), as well as one Deep Learning model, a Temporal Convolutional Network (TCN Bai et al. (2018)).

We used the kNN implementation of the sktime library[6], varying the parameters according to Table 3.3. We use the k-Nearest neighbor as it is one of the standard classification method used for time series. One of the combination used in Table 3.3, 1-NN with the Dynamic Time Warping distance (DTW) is often used in time series analysis (Susto, Cenedese, & Terzi, 2018). STSF is an ensemble of decision trees that samples the training set with class-balanced bagging and creates intervals for 3 representations (the time series, the frequency domain and the derivative representation of the time series) and 7 features (mean, median, standard deviation, slope aggregation functions, inter-quartile range, minimum and maximum). We used the STSF implementation of the sktime library[7] and we varied the number of estimators between 5 and 200. We use this interval-based classifier for its efficiency and also because it is an enhanced method of Random Forest traditionally used in time series classification. STSF is an ensemble of decision trees that samples the training set with class-balanced bagging and creates intervals for 3 representations (the time series, the frequency domain and the derivative representation of the time series) and 7 features (mean, median, standard deviation, slope aggregation functions, inter-quartile range, minimum and maximum). We

[6] https://www.sktime.org/en/stable/api_reference/auto_generated/sktime.classification.distance_based.KNeighborsTimeSeriesClassifier.html

[7] https://www.sktime.org/en/stable/api_reference/auto_generated/sktime.classification.interval_based.SupervisedTimeSeriesForest.html

Parameter	Values tested
Number of neighbours	1, 3, 5, 10, 20, 50
Distance	dtw, ddtw, wdtw, lcss, erp, msm, twe
Weight	uniform, distance

Table 3.3: Parameter combinations tested for the kNN time series classifier. Dtw: dynamic time warping, ddtw: derivative dtw, wdtm: weighted dtw, lcss: longest common sub-sequence, erp: edit distance with real penalty, msm: move split merge, twe: time warping edit

Parameter	Values tested
Kernel size	4
Activation function	softsign, relu, tanh
Number of filters	64
Optimizer	RMSprop, Adam, Adamax, Nadam

Table 3.4: Parameter combinations tested for the TCN time series parameter. RMSprop: root mean square propagation

used the STSF implementation of the sktime library[8] and we varied the number of estimators between 5 and 200. We use this interval-based classifier for its efficiency and also because it is an enhanced method of Random Forest traditionally used in time series classification. For the Temporal Convolutional network, we used the Keras TCN implementation[9] described in Bai et al. (2018), varying the parameters according to Table 3.4. The TCN model used a softmax activation function for its final layer and the sparse categorical cross-entropy loss function. We use the TCN method as it achieves good performance compared to Recurrent Neural Network (RNN) and allows parallel computation for the outputs. We wanted to evaluate the TCN performance for classification tasks in time series as it is mainly used in anomaly detection problems.

To evaluate the classifiers, we split the analysis windows into a training and test sets with a proportion of $70\% - 30\%$. We applied the Grid search cross validation[10] and the Stratified k-Fold[11] iterator from the scikit-learn library to search for the optimal parameters on the training set. The stratify strategy keeps the same class proportion between the folds and the data set. We evaluated

[8] https://www.sktime.org/en/stable/api_reference/auto_generated/sktime.classification.interval_based.SupervisedTimeSeriesForest.html

[9] https://github.com/philipperemy/keras-tcn

[10] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[11] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html

the models using the canonical micro F1 score definition and confusion matrix for each anomaly class.

Chapter 4

Experiments

We conducted experiments to evaluate our anomaly classification method using simulated and real data sets. We also evaluated the performance of the detection method as a sanity check since the performance of the classification model clearly depends on the type of detected anomalies.

4.1 REAL and aREAL data sets preparation

REAL time series Before using the REAL time series, we cleaned the missing parts of the time series (where $\hat{X}(t) = 0$) that usually result from interruption of the monitoring system and are therefore excluded from the anomaly detection model. To clean these time series, we simply deleted the missing parts from the time series. As the TCN needs a continuous time series to fit the model, we rebuilt the missing parts by extracting the weekly seasonality component using STL from the rest of the time series. The missing parts were filled with the mean of o points occurring at the same weekly periodicity time (for example each Monday), where o is an integer between 1 and 3 depending on the size of the sane parts (the maximum of sane occurrence up to 3). We used the STL decomposition for the seasonality extraction as the seasonal component evolve over time compared to the moving average method. We discarded the time series with more than 10% missing parts. We also applied the same sampling period t_s to the REAL time series as well as the filtering and scaling processing in subsection 3.1.3.

We also measured the noise level from each REAL set using a high-pass Butterworth filter with

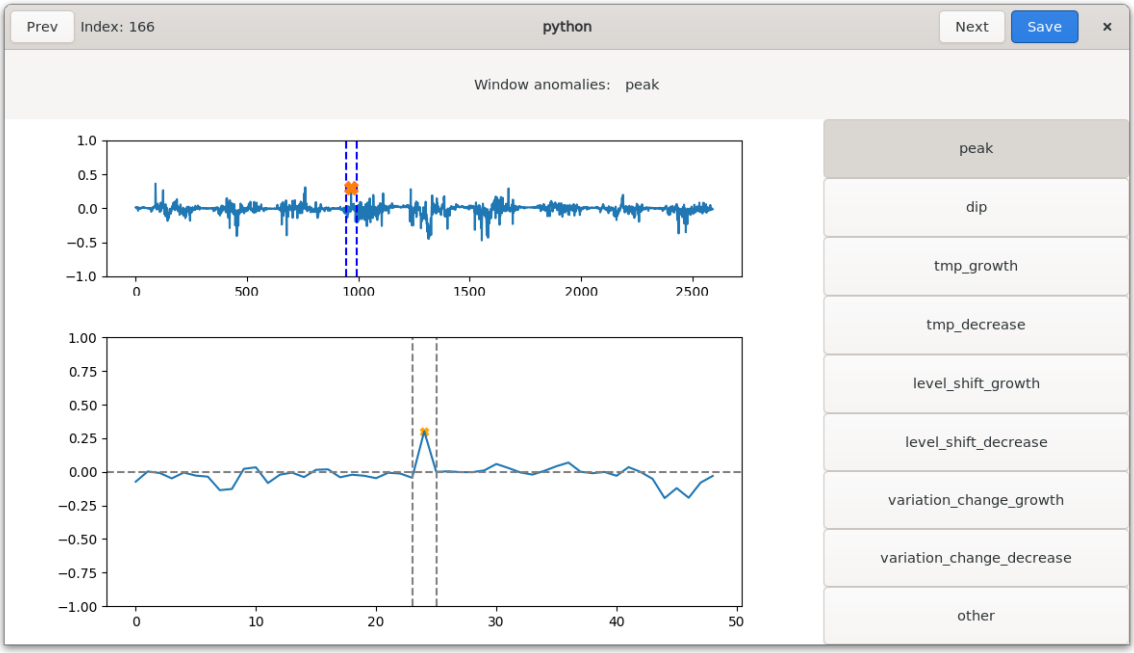


Figure 4.1: Manual classification program

the following parameters:

$$N = 5 \qquad f_s = \frac{1}{t_s \times 60} \qquad W_c = \frac{f_s}{8},$$

where N is the filter order, f_s is the sampling frequency, and W_c the critical frequency and t_s the sampling period defined in Section 3.1. After extracting the high frequencies, we calculated the noise level as the standard deviation of the time series with a degree of freedom equal to 1. The noise level is needed during the experiments to compare results obtained from the same noise level.

aREAL labeling The analysis windows extracted from the real dataset were manually labeled using the classification program shown in Figure 4.1, where the user can assign one or multiple labels for each anomaly detected. The last label "other" is used in case the anomaly cannot be clearly classified into existing classes. The proportion of anomalies classified as "other" represented 0.8% of the total number of analysis windows.

Dataset	Range of noise level σ
aSIM1	$[0.0, 0.01[$
aSIM2 and aREAL1	$[0.01, 0.03[$
aSIM3 and aREAL2	$[0.03, 0.05[$
aSIM4 and aREAL3	$[0.05, 0.07[$
aSIM5	$[0.07, 0.09[$

Table 4.1: Parameter combinations used for the aSIM or aREAL separation

Anomaly class	Subclasses and class proportion		
	Peak or Growth	Dip or Decrease	Total
Single point	0.43	0.02	0.45
Temporary Change	0.38	0.02	0.4
Level Shift	0.005	0.005	0.01
Variation Change	0.1	0.04	0.14

Table 4.2: Proportion of assigned anomaly types in the aREAL dataset

4.2 Datasets

REAL The first data set is composed of the REAL time series with 65 different 1-month-long time series. The 65 time series correspond to the average delay collected from 65 different networks labeled REAL1 to REAL65. Each of these sources has a length of 1 month and may have a different amplitude and seasonality.

aREAL The second data set consists of the aREAL analysis windows created by the Anomaly detection model on the REAL time series. We named the analysis windows from aREAL1 to aREAL3 depending on the range of their noise level (see Table 4.1) as explained previously. We assigned class labels to the analysis windows manually, using the program described previously in Section 4.1, resulting in the class proportions reported in Table 4.2. Most of the detected anomalies were Single point or Temporary change anomalies with the peak or growth direction.

SIM The third data set is the SIM time series generated as detailed in Section 3.1. We extracted the noise level from the REAL time series and the proportion of each class of anomalies from the aREAL analysis windows, and injected them in the simulator. We generated a total of 54 data sets, varying the parameters according to Table 4.3.

Parameter	Values
Number of anomalies n	250, 500, 1000
Noise level σ	0.0, 0.2, 0.4, 0.6, 0.8
Strength α	[0.5, 0.7]
Anomaly proportion	Imbalanced: (0.43, 0.02, 0.38, 0.02, 0.005, 0.005, 0.1, 0.04), Balanced: (0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125, 0.125)

Table 4.3: Parameter combinations used for the SIM dataset creation

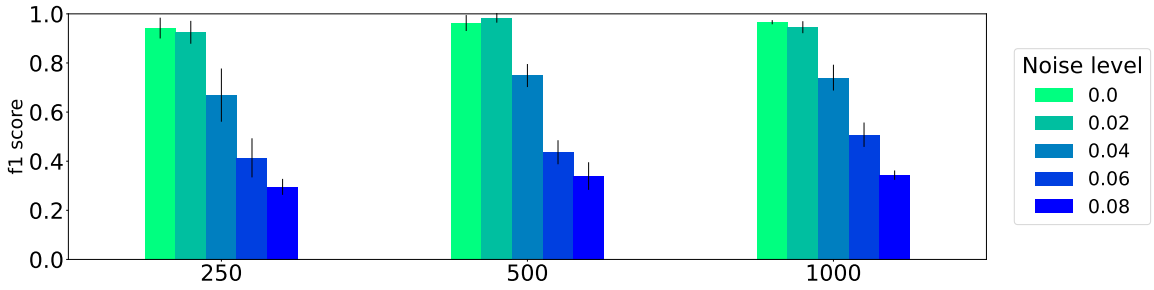
aSIM The fourth data set is composed of the aSIM analysis windows produced by the anomaly detection model on the SIM time series, named aSIM1 to aSIM5 depending on their noise level (Table 4.1).

4.3 Training and evaluation

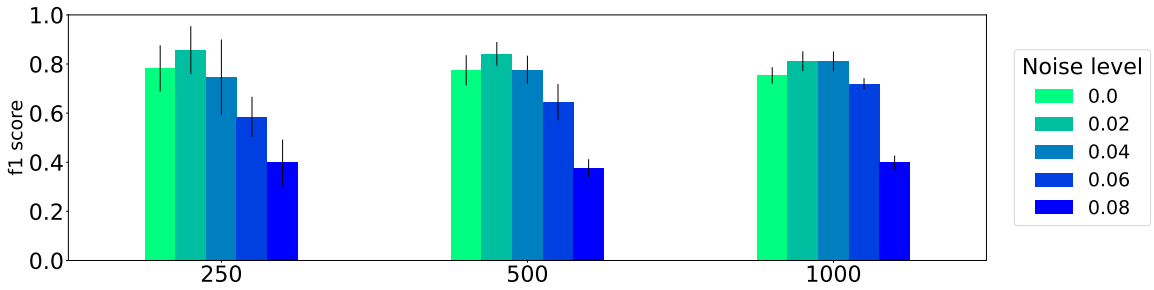
As explained previously, an anomaly can appear in multiple analysis windows and therefore the anomaly proportion may not be the same between the SIM and aSIM sets. For our experiments, we wanted to keep the same proportion between the detection and classification as the imbalanced proportion case is extracted from the aREAL analysis windows. We used two methods to guarantee the same proportion between the anomalies in the SIM and aSIM data sets. In case of a balanced proportion, we downsampled the majority class using the RandomUnderSampler[1] function from the Imbalanced-learn library. In case of an imbalanced proportion, we randomly selected samples from each class according to the anomaly proportion parameter used in the Anomaly Simulation.

SIM-SIM We named the first experiment SIM-SIM as it trained and evaluated the detection and classifiers on the SIM and aSIM data sets, having prepared the aSIM analysis windows to respect the class proportions as detailed before. The goal of this experiment was to evaluate the detection

[1] https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html



(a) Imbalanced data sets



(b) Balanced data sets

Figure 4.2: Detection results, SIM-SIM datasets ($t_s = 5$ minutes)

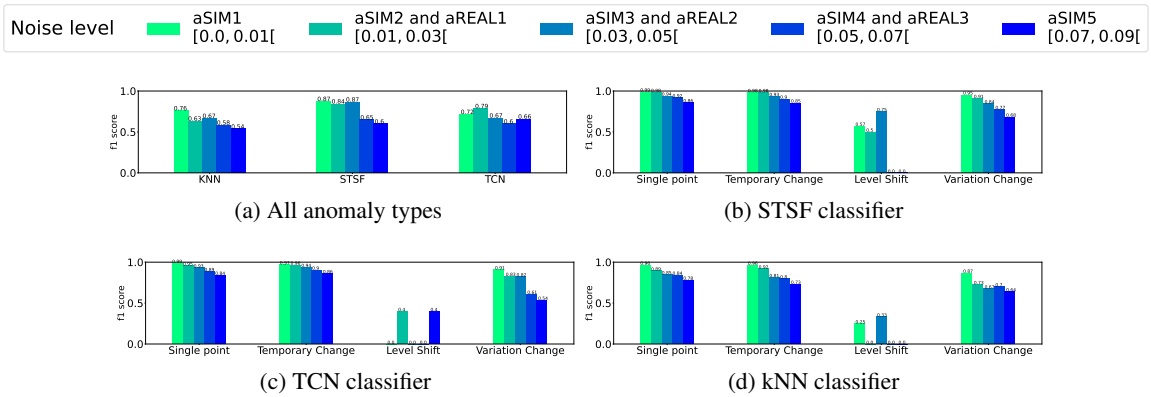
and classification parts on the simulated data sets. In the second experiment, we compared these results to the ones obtained with the REAL data set.

SIM-REAL We named the second experiment SIM-REAL as it trained the classifier on the aSIM dataset and evaluated it on the aREAL dataset. We trained different classifiers for each range of noise level defined in Table 4.1. and we evaluated them independently. The data sets used are aSIM2-4 and aREAL1-3.

4.4 SIM-SIM results

4.4.1 Detection results

Figure 4.2 shows the performance of the detection model for different noise levels. For the imbalanced case, the F1 score decreased as the noise level increased in the SIM dataset $\tilde{ts}$. The balanced case showed an increase until $\sigma = 0.1\mu s$ and then a rapid decrease. The difference in behavior between the imbalanced and balanced dataset reflects the detection of the level shift and



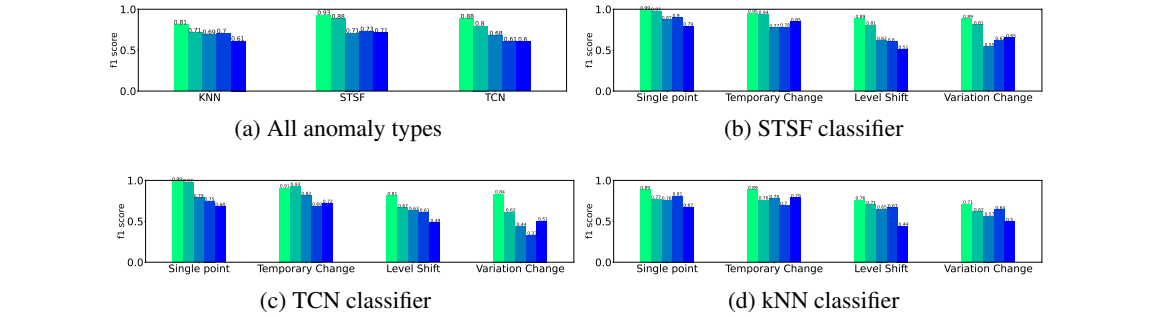
(a) All anomaly types

(b) STSF classifier

(c) TCN classifier

(d) kNN classifier

Figure 4.3: SIM-SIM classification results (imbalanced data)



(a) All anomaly types

(b) STSF classifier

(c) TCN classifier

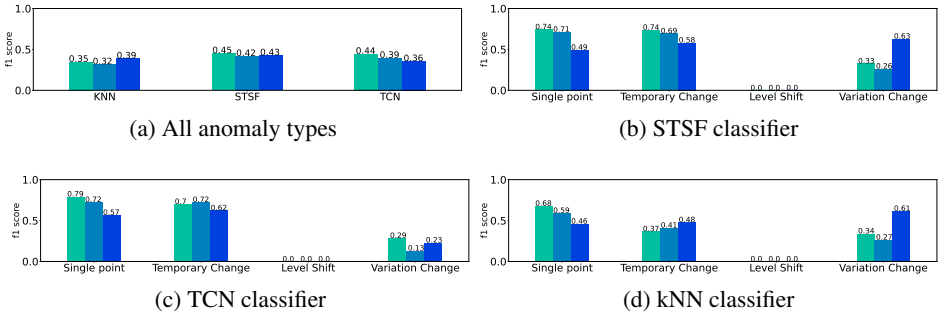
(d) kNN classifier

Figure 4.4: SIM-SIM classification results (balanced data)

variation change classes. These classes are more difficult to detect than the other ones, which explains the reduced performance in the balanced dataset. In the imbalanced dataset, these classes are too infrequent to have a measurable impact on the performance as we used the micro F1 score.

4.4.2 Classification results

Figures 4.3, 4.4 show an overview of the classifiers results for different noise levels and weights. The STSF and TCN models perform the best in both the imbalanced and balanced case. The overall F1 score of the classifiers across all anomaly classes is equal to 0.7 for the imbalanced case and 0.73 for the balanced case. As expected, the F1 score decreases as the noise level increases. We evaluated a decrease of the F1 score around 29% for the TCN and around 23% for the STSF between low (aSIM1) and high noise level (aSIM5). The Single point and Temporary change anomalies have the best overall F1 score.



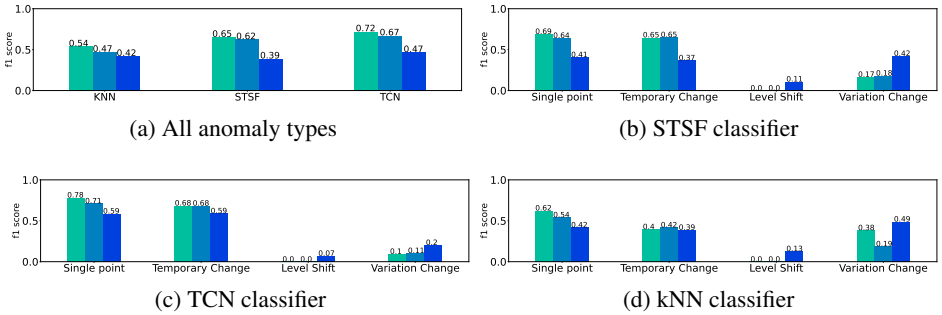
(a) All anomaly types

(b) STSF classifier

(c) TCN classifier

(d) kNN classifier

Figure 4.5: SIM-REAL classification results (imbalanced data)



(a) All anomaly types

(b) STSF classifier

(c) TCN classifier

(d) kNN classifier

Figure 4.6: SIM-REAL classification results (balanced data)

The detection and classification performs well for the simulated data sets. The detection shows good results for both the imbalanced and balanced case with a F1 score higher than chance even for high level of noise.

4.5 SIM-REAL results

As shown in Figure 4.5, 4.6, the STSF and TCN classifiers show good performances for Single point and Temporary change anomalies, even though they were only trained on the simulated dataset. The kNN classifier did not perform as well as the STSF and TCN classifiers. The overall F1 score of the classifiers decreases to 0.39 for the imbalanced case and 0.55 for the balanced case considering all the anomaly types. This relatively poor performance is explained by the difficulty to classify level shifts and variation changes. The overall F1 decrease is due to the insufficient number of anomalies for the Level shift and Variation change types. This type of anomalies are not frequent in the REAL data sets observed. We evaluated a decrease of the F1 score around 24% for the TCN

and around 35% for the STSF between the simulated and real-world datasets.

Chapter 5

Conclusion

5.1 Summary of contributions

The classifiers trained on simulated data (aSIM dataset) achieve relatively good performance (F1 in [0.6-0.93] depending on noise level) when tested on simulated data, which demonstrates the relevance of the approach. When tested on the aREAL datasets, the classifiers achieve a relatively good F1 score (F1 > 0.6) for the single point and temporary change anomalies and for low levels of noise. Under these assumptions, trained models using a simulated approach appears to be transferable to real data processing. This demonstration holds significance and represents the primary value derived from this research endeavor.

The poor classification of level shift and variation change anomalies results from the difficulty to separate these classes on 2-hour anomaly windows, which was observed in the simulated dataset and confirmed in the real dataset. To address this issue, one could adopt larger or adaptive anomaly window sizes. As expected, the noise level also directly impacts the classifiers performance. Pre-filtering the signal might help reducing sensitivity to noise.

5.2 Future work

Multiple improvements could be envisaged in the future. First, the anomaly simulator could support superimposed anomalies that occur frequently in real time series. The simulator could also

modulate the signal amplitude and trend according to variation patterns observed in the real data, such as variations between week days and weekends.

The detection and data preparation steps also have a large impact on the classification. A better integration between data preparation, anomaly detection, and anomaly classification could help improve the performance of the classifier. For instance, we could create a detector for each class of anomalies and optimize their parameters according to it (for example weekly prediction for the Level shift and a hourly prediction for the Single point anomalies). This modification would help classify the anomalies depending on the detector. The decomposition could also be done with the detector prediction as the TCN is able to learn the trend/seasonality of the time series.

Another way could be to create two layers of classification, the first layer could identify the length, trend, amplitude and noise level, given the results from the first layer, we could separate the Single point and Temporary change from the Level shift and Variation change, as they can already be separated by the length. The second layer would be specialized in either classifying Single point and Temporary change or Level shift and Variation change given the information extracted by the first layer.

Another integration could be to give more information about the anomalies such as the approximate length (for example the number of consecutive points detected as an anomaly or the length of the time series which contains around 80-90% or anomalous points), the seasonality, and the trend. This integration could help create variable analysis windows lengths.

Another improvement would be to give multiple outputs and the probability associated for the classifiers. This modification would greatly improve the reliability of the framework by giving a confidence value.

Finally, another improvement could be to directly train the classifiers on real data sets, which would require larger sets of labeled anomalies. We suggested this improvement as the classification shows good performance when trained and tested on the same dataset.

A natural progression of this work would be to allow for variable analysis windows lengths and treating the cases of superposition of different anomaly types. Future studies could also evaluate the anomaly strength detected or characterize even further the anomaly during the classification.

References

Bagnall, A., & Lines, J. (2014). An experimental evaluation of nearest neighbour time series classification. *arXiv preprint arXiv:1406.4757*.

Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data mining and knowledge discovery*, *31*, 606–660.

Bai, S., Kolter, J. Z., & Koltun, V. (2018). An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*.

Bordeau Aubert, K., Whatley, J., Nadeau, S., Glatard, T., & Jaumard, B. (2023). Classification of anomalies in telecommunication network kpi time series.

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2017). Conditional time series forecasting with convolutional neural networks. *arXiv preprint arXiv:1703.04691*.

Box, G. E., & Pierce, D. A. (1970). Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American statistical Association*, *65*(332), 1509–1526.

Cabello, N., Naghizade, E., Qi, J., & Kulik, L. (2020). Fast and accurate time series classification through supervised interval search. In *2020 ieee international conference on data mining (icdm)* (p. 948-953). doi: 10.1109/ICDM50108.2020.00107

Chen, Y., Keogh, E., Hu, B., Begum, N., Bagnall, A., Mueen, A., & Batista, G. (2015, July). *The ucr time series classification archive.* (www.cs.ucr.edu/~eamonn/time_series_data/)

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y.

(2014, October). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)* (pp. 1724–1734). Doha, Qatar: Association for Computational Linguistics. Retrieved from https://aclanthology.org/D14-1179 doi: 10.3115/v1/D14-1179

Choi, K., Yi, J., Park, C., & Yoon, S. (2021). Deep learning for anomaly detection in time-series data: review, analysis, and guidelines. *IEEE Access*, *9*, 120043–120065.

Cleveland, R. B., Cleveland, W. S., McRae, J. E., & Terpenning, I. (1990). Stl: A seasonal-trend decomposition. *J. Off. Stat*, *6*(1), 3–73.

Dagum, E. B., & Bianconcini, S. (2016). *Seasonal adjustment methods and real time trend-cycle estimation*. Springer.

Deng, H., Runger, G., Tuv, E., & Vladimir, M. (2013). A time series forest for classification and feature extraction. *Information Sciences*, *239*, 142–153.

Elman, J. L. (1990). Finding structure in time. *Cognitive science*, *14*(2), 179–211.

Faouzi, J. (2022). Time series classification: A review of algorithms and implementations. *Machine Learning (Emerging Trends and Applications)*.

Foorthuis, R. (2018). A typology of data anomalies. In *Information processing and management of uncertainty in knowledge-based systems. theory and foundations. ipmu 2018. communications in computer and information science* (Vol. 854). Springer, Cham.

Foorthuis, R. (2021). On the nature and types of anomalies: A review of deviations in data. *International Journal of Data Science and Analytics*, *12*(4), 297–331.

Hyndman, R. J., & Athanasopoulos, G. (2018). *Forecasting: principles and practice*. OTexts.

Ismail Fawaz, H., Lucas, B., Forestier, G., Pelletier, C., Schmidt, D. F., Weber, J., ... Petitjean, F. (2020). Inceptiontime: Finding alexnet for time series classification. *Data Mining and Knowledge Discovery*, *34*(6), 1936–1962.

Lea, C., Vidal, R., Reiter, A., & Hager, G. D. (2016). Temporal convolutional networks: A unified approach to action segmentation. In *Computer vision–eccv 2016 workshops: Amsterdam, the netherlands, october 8-10 and 15-16, 2016, proceedings, part iii 14* (pp. 47–54).

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Madan, R., & Mangipudi, P. S. (2018). Predicting computer network traffic: a time series forecasting approach using dwt, arima and rnn. In *2018 eleventh international conference on contemporary computing (ic3)* (pp. 1–5).

Malhotra, P., Vig, L., Shroff, G., Agarwal, P., et al. (2015). Long short term memory networks for anomaly detection in time series. In *Esann* (Vol. 2015, p. 89).

Mei, X., Pan, E., Ma, Y., Dai, X., Huang, J., Fan, F., ... Ma, J. (2019). Spectral-spatial attention networks for hyperspectral image classification. *Remote Sensing*, *11*(8), 963.

Seitzer, M., Tavakoli, A., Antic, D., & Martius, G. (2022). On the pitfalls of heteroscedastic uncertainty estimation with probabilistic neural networks. In *International conference on learning representations*. Retrieved from https://openreview.net/forum?id=aPOpXlnV1T

Susto, G. A., Cenedese, A., & Terzi, M. (2018). Time-series classification methods: Review and applications to power systems data. *Big data application in power systems*, 179–220.

Torres, J. F., Hadjout, D., Sebaa, A., Martínez-Álvarez, F., & Troncoso, A. (2021, February). Deep Learning for Time Series Forecasting: A Survey. *Big Data*, *9*(1), 3–21. Retrieved 2023-01-26, from https://www.liebertpub.com/doi/10.1089/big.2020.0159 doi: 10.1089/big.2020.0159

Xue, N., Triguero, I., Figueredo, G. P., & Landa-Silva, D. (2019). Evolving deep cnn-lstms for inventory time series prediction. In *2019 ieee congress on evolutionary computation (cec)* (pp. 1517–1524).

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