TÉCNICO LISBOA

Descriptive and Predictive Modeling of Water Distribution Network Dynamics using Multivariate Time Series Data

Susana Chambel Gonçalves Correia Gomes

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor(s): Prof. Rui Miguel Carrasqueiro Henriques
Prof. Susana de Almeida Mendes Vinga Martins

Examination Committee

Chairperson: Prof. Nuno João Neves Mamede
Supervisor: Prof. Rui Miguel Carrasqueiro Henriques
Member of the Committee: Prof. Alexandre Paulo Lourenço Francisco

January 2021

For my grandparents

Acknowledgments

I would like to express my sincere gratitude to my two amazing supervisors Prof. Rui Henriques and Prof. Susana Vinga for their valuable time and support during my journey in Instituto Superior Técnico. I also want to thank Carolina, Lourenço, and Miguel for making these two years way more fun and memorable. My academic journey would have also not been the same if it wasn't for Rodrigo, the best working partner and friend I could have wished for. I also want to thank Diana for keeping me sane during this pandemic and for always making me smile. I am also forever grateful to Ângela, Simona, and Tapadas for being a constant in my life. Lastly, I want to thank my lovely parents and grandparents for always supporting me and believing in me more than myself.

This work was developed in the context of the WISDOM project (DSAIPA/DS/0089/2018) and supported by national funds through *Fundação para a Ciência e Tecnologia* under a three-month grant from ILU project (DSAIPA/DS/0111/2018).

Resumo

Redes de distribuição água (RDAs) são infraestruturas hidráulicas responsáveis por aprovisionar um abastecimento contínuo e pressurizado de água potável, tendo assim um papel essencial na saúde pública. No entanto, a presença de fugas desperdiça recursos e compromete a qualidade da água. Embora existam métodos que apoiam a monitorização e controlo de RDAs, estes demonstram uma capacidade limitada na deteção de anomalias, não sendo ainda aplicados de forma consistente em RDAs portugueses. Neste trabalho, mostramos que é possível (1) descrever as dinâmicas de uma RDA através da análise de correlações espaciotemporais de sensores de pressão e caudal, e (2) analisar as disrupções nestas correlações para detetar dinâmicas de fuga com recurso a classificadores. Esta abordagem revelou-se promissora em ambiente simulado e, apesar dos desafios encontrados, apresenta uma solução inicial que suporta a deteção em ambiente real. Descobrimos ainda que a interrupção causada pelas fugas é maior logo após a sua ocorrência. Além disso, a seleção de pares de sensores e balanceamento dos dados em ambiente real são também promissores. Os resultados sugerem que é importante ter acesso a dados provenientes de uma boa rede de sensores e a informação completa sobre as fugas. Consequentemente, acreditamos que a RDA estudada beneficiaria muito com a expansão e realocação dos sensores. Por último, dada a simplicidade, novidade e precisão dos princípios de correlação propostos para a deteção de anomalias em series temporais heterogenias e georreferenciadas, antecipamos que o nosso trabalhado irá contribuir para o estudo da deteção automática de fugas em RDAs portuguesas.

Palavras-chave: Análise de Correlações, Analise de Dados Espaciotemporais, Deteção de Fugas de Água, Previsão, Rede de Distribuição de Água, Séries Temporais Multivariadas.

Abstract

Water distribution networks (WDNs) are hydraulic infrastructures that provide a continuous supply of pressurized safe drinking water to all consumers, playing an important role in public health. Leakages cause service interruptions, waste resources, and compromise water quality. Although we can find many methods to support the monitoring and control of WDNs, they exhibit limited ability to detect anomalies and are not yet consistently applied to Portuguese WDNs. We show that it is possible to (1) describe the dynamics of a WDN through spatiotemporal correlation analysis of pressure and volumetric flowrate sensors, and (2) analyze disruptions on the expected correlation to detect burst leakage dynamics using standard classifiers. Our approach is promising in a synthetic setting and offers initial support towards leakage detection in real WDNs despite the presence of highly irregular consumption patterns, a limited number of recorded leakages, and highly heterogeneous leakage profiles. We discovered that the disruption caused by leakages is higher shortly after the burst. Furthermore, a comprehensive pairing of heterogeneous sensors and data balancing in the real setting is also promising. Our results suggest that it is important to access data from WDNs with good sensor coverage and complete information about leakages. Accordingly, we believe that our WDN would benefit a lot from sensor expansion and relocation. Lastly, given the simplicity, novelty, and accuracy of the proposed correlation-based principles for anomaly detection in heterogeneous and georeferenced time series, we anticipate our work to contribute to the study and development of automated leakage detection in Portuguese WDNs.

Keywords: Burst Leakage Detection, Correlation Analysis, Multivariate Time Series, Prediction, Spatiotemporal Data Analysis, Water Distribution Network.

Contents

List of Tables

List of Figures

Acronyms

ANOVA Analysis of Variance

ARIMA Autoregressive Integrated Moving Average

AUC Area Under the ROC Curve

DCCA Detrended Cross-Correlation Analysis

DDTW Derivative Dynamic Time Warping

DFA Detrended Fluctuation Analysis

DTW Dynamic Time Warping

EDR Edit Distance on Real Sequences

FN False Negatives

FPR False Positive Rate

FP False Positives

ID Database Identifier

M-SSA Multi-Channel Singular Spectrum Analysis

NB Naive Bayes

PAA Piecewise Aggregate Approximation

PCA Principal Component Analysis

PCC Pearson's Cross-Correlation Coefficient

PDTW Piecewise Dynamic Time Warping

RBF Gaussian Radial Basis Function

ROC Receiver Operating Characteristics

SEATS Signal Extraction in ARIMA Time Series

SSA Singular Spectrum Analysis

STL Seasonal-Trend Decomposition Procedure Based on Loess

SVM Support Vector Machine

TN True Negatives

TPR True Positive Rate

TP True Positives

TRAMO Time Series Regression with ARIMA Noise, Missing Observations, and Outliers

TWED Time-Warped Edit Distance

UMAP Uniform Manifold Approximation and Projection for Dimension Reduction

WDN Water Distribution Network

WISDOM Water Intelligence System Data

WU Water Utility

Chapter 1

Introduction

Water distribution networks (WDNs) are hydraulic infrastructures responsible for providing a pressurized and continuous supply of safe drinking water to all consumers, using hydraulic components, such as pipes, valves, and reservoirs [1]. These networks have an important role in public health and community growth, and with climate change, affluence, and population growth demanding higher consumption levels of water, the need for well-managed systems is more important than ever [2, 3]. Moreover, the presence of leakages can cause service interruptions, contribute to resource wastes, and increase the risk of compromising water quality [4]. Therefore, it is vital to make WDNs more efficient by providing proper monitoring and control.

Although we can find many methods and algorithms that improve the monitoring and control of WDNs, such as WaterWiSe [5] in Singapore, they show limited ability to detect anomalies and are not yet consistently applied to Portuguese WDNs. Newly approved projects, such as the water intelligence system data (WISDOM), are addressing these challenges [6]. WISDOM intends to perform knowledge discovery through data collected by Portuguese water utilities (WUs), namely, Municipality of Barreiro, Municipality of Beja, and Infraquinta. Additionally, despite the existing efforts, we did not find any studies that comprehensively modeled the spatiotemporal dynamics in a WDN from the relationships between sensors and use its disruption to detect anomalous behavior and other events of interest.

1.1 Problem Description

Considering the previous context, we will study the spatiotemporal correlations of pressure and volumetric flowrate sensors under normal conditions to detect anomalous behavior when the normal conditions suffer a disruption. Therefore, and as part of the WISDOM project, our main focus is to detect burst leakages according to Portuguese necessities by identifying descriptors and predictors of WDN dynamics. Due to the nature of the problem, we identified several challenges that may hamper the performance of the description and prediction tasks:

- Poor sensor coverage;
- Highly irregular consumption patterns;

- Leakages can have multiple profiles, such as size and location;
- Low number of leakages and lack of information regarding its size and exact beginning time point;
- Network changes and interventions that disrupt the natural behavior of the network;
- The necessity of detecting leakages as soon as possible.

Additionally, to identify the descriptors of dynamics, we had to explore and analyze high volumes of data that contained gaps and irregular time steps of recording. The data comprised numerous multivariate time series from volumetric flowrate and pressure sensors placed throughout WDNs of Municipality of Barreiro, Municipality of Beja, and Infraquinta. We also had to carry out several experiments in a controlled setting as an attempt to make our predictors overcome the challenges presented earlier.

1.2 Contributions

Through the development of a new technique that uses the disruption of spatiotemporal correlations between sensors to detect burst leakages in WDNs, we provide the following contributions:

- Comprehensive analysis of network dynamics in a real WDN versus in a synthetic WDN;
- Assessment of two correlation methods as descriptors of network dynamics;
- Study of how the correlation between sensors evolves over time;
- Performance assessment of three standard classifiers on leakage detection;
- Study of the importance of feature/sensor selection in leakage detection;
- Study of the impact of the leakage size (flowrate) in its detection.

These contributions could offer direct support to answer alternative problems in WDNs:

- Optimal sensor placement through the installation of new sensors where current sensor correlations are weak;
- The detection of status' changes in active hydraulic elements, since the opening and closure of valves might cause the disruption of correlations between sensors.

1.3 Thesis Outline

The remainder of this thesis is organized as follows:

- Chapter 2 introduces the fundamental topics of this work, i.e., concepts related to time series data and water distribution networks;
- Chapter 3 provides an overview of the existing literature on topics related to our challenges, namely, time series preprocessing and relationship analysis, feature extraction, burst leakage detection techniques, and an overview of existing contributions on the secondary applications of our work;

- Chapter 4 describes the solution outlined according to the topics and literature explored previously;
- Chapters 5 and 6 present a comprehensive analysis of the steps performed to identify the descriptors of network dynamics and the most suitable leakage dynamic predictors for Infraquinta's WDN, using a synthetic and a real setting;
- Chapter 7 details the visualization tool developed to support the analysis of time series data in the context of WDNs;
- Chapter 8 summarizes the main conclusions of this work and provides future directions.

Chapter 2

Background

This chapter introduces the fundamental topics of this work, which we separated into two sections. Section 2.1 comprises of technical concepts, more specifically, the ones related to time series data, and section 2.2 involves concepts associated with this thesis' domain, i.e., water distribution networks.

2.1 Technical Background

Since time series data is our main source of information, we start by defining a time series and explaining its components. Additionally, we explore the challenges of time series classification, as well as existent solutions.

2.1.1 Time Series Definitions

A discrete time series is described as a sequence of T observations, each one being measured at a discrete time $t \in \{1, \ldots, T\}$, made sequentially and regularly along T instances of time. A real-valued observation of a time series, $\mathbf{x}_t = (x_{1t}, \ldots, x_{kt})$, where k is the multivariate order, $t \in \{1, \ldots, T\}$, and $x_{kt} \in \mathbb{R}$ [7]. A time series is called univariate when $k = 1$, and multivariate when $k > 1$.

Time series can also be decomposed into a set of non-observable (latent) components associated with different types of temporal variations. These types of seasonal characteristics are especially relevant in the present context of WDNs due to the inherent period trends present. Persons [8] was the first to define a meaningful group of time series' components into (1) a long-term tendency or secular trend, (2) cyclical movements superimposed upon the long-term trend, (3) a seasonal movement within each year, the shape of which depends on the nature of the series, and finally (4) residual variations due to changes impacting individual variables or other major events. Moreover, if these components are assumed to be mutually independent of one another, their relationship is specified through an additive decomposition model. On the other hand, when in the presence of scaling dependencies, a multiplicative decomposition model is used. It is also important to note that some series can be affected by other variations associated with the composition of the calendar. For example, we can consider trading day variations since some days of the week might be more important than others. Considering that some holidays change the date

from year to year (e.g., Easter), we can also use the moving holiday component [9].

2.1.2 Time Series Classification

Classification is the process of predicting a class label for a given unlabeled instance [10]. The prediction depends on the classifier, which uses training data to learn and create rules to classify new instances. Classical classifiers, such as the naive Bayes classifier and support vector machines, assume that features are not dependent on the ordering [11]. Consequently, these classifiers will handle time series' time points as separate features. Several algorithms consider time series as a whole, i.e., without ignoring feature ordering. For example, the one nearest neighbor classifier with an Euclidean distance or with a dynamic time warping (DTW) is usually the starting point for most researchers [11].

According to Bagnall et al. [11], we can group time series classifiers into six categories. First, time domain distance based classifiers use distance metrics, such as the DTW, to determine the time series' class.

In a second category, we have differential distance based classifiers, which combine distances in the time domain and the difference domain. Dictionary based classifiers start by transforming the time series into a sequence of discrete words and then compare its distribution. The following category regards shapelet based classifiers. Shapelets are subsequences of a time series that characterize a class, thus allowing the detection of phase-independent localized similarities. Then, we have interval based classifiers which extract features, such as the mean and standard deviation, from time series' intervals. Lastly, ensemble classifiers use multiple classifiers, like the ones previously described, to determine the time series' class. This diversity allows ensemble approaches to be highly competitive with general classification problems.

2.1.3 Classical Classification

Extracting features from time series' intervals allows the creation of a new dataset without explicit feature ordering, making classical classifiers a viable solution. Note that this work does not intend to present a comparative study of different classifiers. We only want to assess the potential of the extraction of features from time series in the context of WDNs, focusing only on two well-known classifiers, namely, the naive Bayes classifier and support vector machines. A comprehensive review of best-known classification techniques can be found in Kotsiantis et al. [12].

Naive Bayes Classifier

The naive Bayes (NB) classifier uses a probabilistic classification approach that simplifies learning by assuming all features are conditionally independent given the class [10]. First, consider a training dataset with m points $\mathbf{x}_i \in R^d$ in a d-dimensional space, where $i \in \{1, \ldots, m\}$, y_i is the class for each point, with $y_i \in \{c_1, c_2, \cdots, c_k\}$. Additionally, $\mathbf{D}_i$ represents the subset of points that are labeled as class c_i, and m_i its size. Accordingly, the Bayes classifier uses the Bayes theorem to predict the class of $\mathbf{x}$ as the one that

maximizes the posterior probability $P(c_i \mid \mathbf{x})$,

$$\hat{y} = \arg\max_i \{P(c_i \mid \mathbf{x})\}. \tag{2.1}$$

The posterior probability of class c_i,

$$P(c_i \mid \mathbf{x}) = P(\mathbf{x} \mid c_i) P(c_i), \tag{2.2}$$

is the product between the likelihood $P(\mathbf{x} \mid c_i)$ and the prior probability $P(c_i)$ of class c_i. Essentially, the posterior probability of class c_i depends on the likelihood of that class taking its prior probability into account. The prior probability of class c_i is defined as

$$P(c_i) = \frac{m_i}{m}, \tag{2.3}$$

while the likelihood is defined as the probability of observing $\mathbf{x}$ assuming that the true class is c_i. Due to the initial assumption that all features are independent given the class, we can decompose the likelihood of class c_i into a product of the likelihood along each dimension X_j,

$$P(\mathbf{x} \mid c_i) = P(x_1, x_2, \cdots, x_d \mid c_i) = \prod_{j=1}^{d} P(x_j \mid c_i). \tag{2.4}$$

When a numeric feature is shown to be approximately normally distributed for a given class c_i, the likelihood for class c_i, for variable X_j, is given as

$$P(x_j \mid c_i) = \frac{1}{\sqrt{2\pi}\sigma_{ij}} \exp\left\{-\frac{(x_j - \mu_{ij})^2}{2\sigma_{ij}^2}\right\}, \tag{2.5}$$

with μ_{ij} and σ_{ij}^2 as the mean and variance for variable X_j, for class c_i.

Although feature independence is usually violated in real datasets, Rish et al. [13] shows that NB is a very competitive classifier in diverse application domains and works best when the data either has completely independent features or has functionally dependent features.

Support Vector Machines

Support vector machines (SVMs) are a classification method that sees observations in a d-dimensional space and finds a hyperplane that maximizes the gap or margin between the observations from different classes [10]. In a two dimensional space, this optimal hyperplane corresponds to a line that separates the classes. When it is not possible to find a hyperplane in the current dimension that separates the classes completely, we can use the kernel trick to find an optimal hyperplane in a high-dimensional space. It is also important to note that compared to other methods, SVMs are complex enough to be successfully used in real-world applications while also being simple enough to be analyzed mathematically [14].

Assuming a dataset $\mathbf{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ with m points in a d-dimensional space and with two class labels $y_i \in \{+1, -1\}$. An hyperplane serves as a linear discriminant, i.e., a decision boundary that predicts the

class y for any point $\mathbf{x}$. An hyperplane $h(\mathbf{x})$ in d dimensions is defined as

$$\begin{aligned} h(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} + b \\ &= w_1x_1 + w_2x_2 + \cdots + w_dx_d + b, \end{aligned} \tag{2.6}$$

where $\mathbf{w}$ is a d-dimensional weight vector, and b is a scalar called bias. The hyperplane $h(\mathbf{x})$ splits the original d-dimensional space into two half-spaces. When each half-space has points from a single class, we call it a separating hyperplane [10].

Considering that the distance of a point $\mathbf{x}$ from the hyperplane $h(\mathbf{x}_i) = 0$ is given by

$$\delta_i = \frac{y_i\ h(\mathbf{x}_i)}{\|\mathbf{w}\|}, \tag{2.7}$$

support vectors are all data points that achieve the minimum distance,

$$\delta^* = \min_{\mathbf{x}_i} \left\{ \frac{y_i\ h(\mathbf{x}_i)}{\|\mathbf{w}\|} \right\}. \tag{2.8}$$

Therefore, support vectors are the points closer to the hyperplane and their distance from it is called margin. These points, or vectors, influence the position and orientation of the hyperplane and are chosen to maximize the margin of the classifier.

We will now transform our hyperplane into a canonical hyperplane by choosing a scalar s that guarantees that the absolute distance of a support vector from the hyperplane is 1,

$$s \times y^* \left(\mathbf{w}^T\mathbf{x}^* + b\right) = 1, \tag{2.9}$$

which results in

$$y_i \left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1, \text{ for all points } \mathbf{x}_i \in \mathbf{D}. \tag{2.10}$$

With this in mind, we can write the margin of a canonical hyperplane as

$$\delta^* = \frac{y^* h\left(\mathbf{x}^*\right)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}. \tag{2.11}$$

Therefore, the main idea of SVMs is to find the canonical hyperplane with the maximum margin among all possible separating canonical hyperplanes,

$$h^* = \arg\max_h \{\delta_h^*\} = \arg\max_{\mathbf{w},b} \left\{ \frac{1}{\|\mathbf{w}\|} \right\} \tag{2.12}$$

Figure 2.1 depicts a separating canonical hyperplane $h(\mathbf{x}) = 0$ where the triangles and circles are points from two different classes, and the shaded ones are the support vectors.

Since not all datasets are linearly separable, we can introduce slack variables ξ_i in (2.10), resulting in

$$y_i \left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 - \xi_i. \tag{2.13}$$

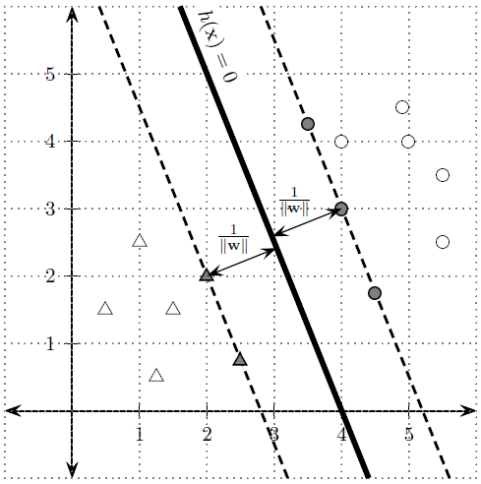

Figure 2.1: An illustrative separating canonical hyperplane [10].

We call this case the soft margin case, which goal is not only to find the hyperplane with the maximum margin but one that also minimizes the slack terms [10].

Another way to solve the non-separable cases is to use the kernel trick. The idea is to map the original d-dimensional points in a high-dimensional space, called feature space, via non-linear transformations, in hopes that the data will be linearly separable there. It is also important to note that this trick allows us to find the optimal hyperplane without actually having to map the points in the new space. Instead, kernel functions find the non-linear relationships among features [10]. One popular function is the polynomial kernel, which is defined as

$$K_q(\mathbf{x}, \mathbf{y}) = \left(c + \mathbf{x}^T \mathbf{y}\right)^q, \tag{2.14}$$

where $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, q is the degree of the polynomial and $c \geq 0$ is a constant. The linear and quadratic kernels are special cases of the polynomial kernel when $q = 1$ and $q = 2$, respectively. The Gaussian radial basis function (RBF) is another well known kernel function and is defined as

$$K(\mathbf{x}, \mathbf{y}) = \exp\left\{-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2}\right\}, \tag{2.15}$$

where $\sigma > 0$ is a constant. It is important to note that the RBF kernel has infinite dimensionality. According to Hofmann [15], a low polynomial kernel or an RBF kernel are good alternatives to conventional classifiers.

2.2 Water Distribution Network Background

In this section, we start by introducing essential aspects of water distribution networks and explaining what leakages are and how they are related to total water losses of a network. Then, we show two different systems that water utilities use to maintain their networks. Finally, we overview applications that model water distribution networks, allowing the generation of synthetic data.

2.2.1 Water Distribution Networks

Water distribution networks (WDNs), as the one represented in Figure 2.2, are hydraulic infrastructures that provide a continuous supply of pressurized safe drinking water to all consumers [1]. These networks

Figure 2.2: Overview of Infraquinta's WDN.

are generally composed of a large number of active and passive hydraulic elements. The active elements, such as pumps and valves, can be operated to control the flowrate and pressure of water in specific sections of the network. Passive elements, such as pipes and reservoirs, receive the effects of the operation of active elements [16]. Table 2.1 lists the most common hydraulic elements in a WDN and their primary purpose.

Element	Type	Primary Purpose
Reservoir	Passive	Provides water to the system.
Tank	Passive	Stores excess water within the system and releases that water at times of high usage.
Junction	Passive	Provides a location for two or more pipes to meet, removing (demand) or adding (inflow) water from/to the system.
Pipe	Passive	Conveys water from one junction node to another.
Pump	Active	Raises the hydraulic grade to overcome elevation differences and friction losses.
Control	Active	Controls flowrate or pressure in the system based on specified criteria.

Table 2.1: Common hydraulic elements in a WDN [16, 17].

2.2.2 Water Pressure and Volumetric Flowrate

The pressure is described as the normal force per unit area at a given point acting on a given plane within the water mass of interest. Regarding the volumetric flowrate, simply referred to as flowrate in this work, corresponds to the amount of water flowing in a certain point per unit of time [18]. It is also worth noting the flowrate is proportional to the square root of the pressure difference between two points. An effective way to measure it through a pipe is to restrict it, as shown in Figure 2.3. We can now represent the

theoretical flowrate,

$$Q = A_2 \sqrt{\frac{2\,(p_1 - p_2)}{\rho\left[1 - (A_2/A_1)^2\right]}}, \tag{2.16}$$

where p_1 and p_2 correspond to the pressure at sections (1) and (2) in Figure 2.3, respectively, ρ is the density of the flowing fluid, and A_2 is the smallest ($A_2 < A_1$) flow area at section (2) in Figure 2.3. Due to the differences between the real world and the assumptions used in the derivation of the equation, the measured flowrate will be smaller than the theoretical result of Eq. (2.16) [18]. Throughout this work, we use *bar* and m^3/h as measurement units for the pressure and volumetric flowrate, respectively.

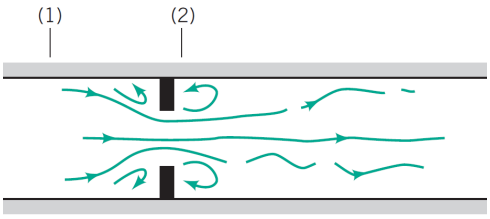

Figure 2.3: Simplification of a device for measuring the volumetric flowrate in pipes [18].

2.2.3 Water Loss and Leakages

The total water loss is described as the difference between the amount of water produced and the amount which is billed or consumed [19]. Water losses can be categorized as (1) non-physical (or apparent) losses, which are a result of unauthorized consumption and metering inaccuracies, and (2) physical (or real) losses, which include leakages from pipes, joints and fittings, leakages through reservoir floors and walls, and also leakages from reservoir overflows [20].

Leakages are also commonly defined as the loss of treated water from the network through uncontrolled means [19, 21]. They are categorized in literature as (1) background leakages, which consist of the aggregation of leakages small enough to be undetected for long periods of time, and (2) burst leakages, which are defined as occurring pipe ruptures that usually result in a large water discharge [20]. We can model a leakage based on the equation of flowrate through an orifice,

$$q_j = K_j p_j^{\beta}, \tag{2.17}$$

where q_j is the leakage flowrate at node j, p_j the pressure at node j, and β the pressure exponent. K_j is a fixed leakage coefficient for the node and was estimated as a function of pipe and soil characteristics,

$$K_i = c \times \sum_{J=1}^{M} 0.5 \times L_{ij}, \tag{2.18}$$

where c is the discharge coefficient of the orifice which depends on the shape and the diameter, L_{ij} is the pipe length between nodes i and j, and finally, M is the number of pipe reaches connected to the node j [22].

2.2.4 Water Utilities

Water utilities (WUs) are public or private entities responsible for the management of WDNs. We have three WUs involved in this work, namely, Municipality of Barreiro, Municipality of Beja, and Infraquinta. To facilitate the management of their networks, these WUs use two systems, more precisely, a (1) telemanagement and a (2) telemetry system.

The (1) telemanagement system allows the remote management of WDNs, including the control of production processes, data collection, and generation of information through analysis, graphs, and reports. This information is available in a command center, allowing WUs to make operational decisions under the current status of the system [23]. Regarding the (2) telemetry system, it allows the remote transmission and measurement of network data [24]. This system is mostly used for billing purposes, i.e., to assess the exact amount of water a client consumed.

Finally, both systems rely on the existence of sensors and monitoring devices placed throughout the network. These devices respond to a physical stimulus and transmit a resulting measurement, for example, water pressure and volumetric flowrate [25]. The real data considered in the present thesis was measured by multiple sensors and later retrieved from the two previously mentioned systems.

2.2.5 Artificial Water Distribution Network Models

As a result of regulatory requirements and customer expectations, the need for WUs to understand the paths and transformations of treated water in WDNs required the creation of artificial WDN models [26]. Given the shortage of real data to support such analysis, a variety of paid and public modeling software programs that allow the generation of synthetic data in all points of WDNs is now available.

EPANET, for example, is a public domain software created by the United States Environmental Protection Agency that simulates hydraulic and water quality behavior within pressurized pipe networks [26]. EPANET has a variety of applications, such as sampling program design, hydraulic model calibration, chlorine residual analysis, and consumer exposure assessment. Moreover, it also provides an environment for editing network input data, running simulations, and different methods to visualize the results. In the present thesis, all the synthetic data used was generated using an EPANET model of the network.

A range of paid software for WDN modeling is also available. It includes WaterCAD, HydrauliCAD, H2Onet, H2Omap, and Synergi Water, among others. They offer great flexibility over public domain software, and some can even be integrated with other types of software like GIS and SCADA [27].

Chapter 3

Related Work

In this chapter, we overview the existing literature on six different topics. First, section 3.1 introduces principles of time series data preprocessing. Since we need to find descriptors of network dynamics, section 3.2 presents methods responsible for measuring the relationship between time series. Also related to the descriptors, section 3.3 delves into feature extraction. Next, section 3.4 explores different techniques used in literature for burst leakage detection. Finally, in section 3.5, we included an overview of existing contributions on the secondary applications of our work.

3.1 Time Series Data Preprocessing

The data preprocessing step is responsible for catching and solving problems in the data. Although this step often takes a considerable amount of time, skipping it can lead to misleading results [28]. Since time series often suffer from gaps, irregular time steps of recording, or removed data points, that need to be filled, the first concerns reconstruction methods [29]. The other topic refers to time series decomposition, which allows the analysis of specific components of the data and the discovery of consumption patterns and trends related to the calendar's structure.

3.1.1 Reconstruction

Although there is a lack of comprehensive comparative studies on time series reconstruction methods, we introduce some of the available possibilities. Considering observations $\mathbf{x}_A$, $\mathbf{x}_B$ and $\mathbf{x}_C$, that occur in t_A, t_B and t_C, respectively, where $t_A < t_B < t_C$, a very simple and useful approach is the linear interpolation method,

$$\mathbf{x}_B = \frac{\mathbf{x}_A - \mathbf{x}_C}{t_A - t_C}(t_C - t_B) - \mathbf{x}_C. \tag{3.1}$$

Although interpolation yields good results in many situations, it strongly depends on the distribution and characteristics of data [29].

Kondrashov and Ghil [30] propose a solution using singular spectrum analysis (SSA) for univariate time series and multi-channel singular spectrum analysis (M-SSA) for multivariate time series. For univariate records, they use a time lag-covariance matrix, thus, considering temporal correlations. Regarding

multivariate records, the time lag-covariance matrix also considers all channels of the time series, as detailed in Ghil et al. [31]. Therefore, allowing the covariance matrix to reflect both temporal and spatial correlations. Reflecting both correlations enables M-SSA to reconstruct channels (features) through their correlation with others. So, we could use the correlations between flow and pressure to fill in gaps in both flow and pressure measurements separately. Moreover, this method also includes a time window, allowing the algorithm's adaptation to the gaps' length. It is also important to note that the algorithms' accuracy depends on the pattern of missing data and the length of the gaps. Furthermore, although SSA based methods demand high computational requirements, SSA itself proves to be suitable for time series with highly anharmonic oscillation shapes and nonlinear trends [31–33].

Musial et al. [32] adapted an extension of the Lomb-Scargle periodogram that reconstructs a time series using its most relevant spectral components, i.e., amplitude and phase information of the dominant frequencies [34], and that also supports time series with irregular steps of recording. Although the method obtained good results in periodic time series, it has difficulties in dealing with aperiodic ones. It is also important to note its sensitivity to strong trends present in the data. Therefore, the authors advise the removal of any trend before applying the algorithm. Lastly, the article highlights its capability in detecting significant frequencies in arbitrary time series, since this property could be complementary used for forecasting.

Smoothing techniques can also be used to address the reconstruction problem [32, 35]. Eilers [36] presents an extension of the Whitaker smoothing algorithm [37] based on penalized least squares. Although this method proves itself to be trustworthy in many situations, it shows unreliable results when in the presence of medium and large periods of missing data [35]. It is also important to note that this method, as opposed to Kondrashov and Ghil [30], does not use a time window. Nevertheless, this adaptation allows continuous control over smoothness and an automatic interpolation. The authors also highlight the algorithm's easy implementation and high speed.

3.1.2 Decomposition

Throughout the years, several authors have proposed different time series decomposition methods to characterize the underlying structure of a (multivariate) time series. Macauley's approach [38], commonly referred to as classical decomposition, laid the foundations for many modern-day techniques. However, this approach offers some drawbacks, such as assuming the seasonal component repeats from year to year. The estimate of the trend-cycle is also unavailable for the first few and last few observations. Moreover, rapid rises and falls also tend to be over-smoothed. Finally, this method proves itself not robust to outliers [39].

X-11 is a decomposition method developed by the Bureau of the Census [40] and, although based on Macauley's approach, it is much more complex and versatile. This method can handle different situations, including the trading day effect, holiday effect, and extreme observations. Other important features of X-11 include the options of a variety of moving averages for estimating evolving trend and seasonal components, using refined asymmetric moving averages toward the ends of a time series, and many diagnostic statistics to assess the appropriateness of the seasonal adjustment [41]. Additionally,

trend-cycle estimates are available for all observations, including the endpoints. The seasonal component is also allowed to vary slowly over time [39].

Maravall and Gomez [42] proposed an autoregressive integrated moving average (ARIMA) model-based approach called TRAMO/SEATS. ARIMA(p, q, d) is a process that applies an ARMA(p, q) model to a d differenced data, $w_t = \nabla^d x_t$, where x_t is the original non-stationary time series, w_t is the stationarized time series, and $\nabla z_t = z_t - z_{t-1}$ [43]. The ARMA(p, q) model itself is composed of two models, namely, (1) an autoregressive model, AR(p), responsible for relating current observations to previous ones, and (2) a moving average model, MA(q), responsible for correcting the values of the auto-regression by considering the effects of past and present noise terms. Therefore, we define the ARMA(p, q) model,

$$w_t = \phi_1 w_{t-1} + \phi_2 w_{t-2} + \cdots + \phi_p w_{t-p} + \epsilon_t - \theta_1 a_{t-1} - \theta_2 \epsilon_{t-2} - \cdots - \theta_q \epsilon_{t-q}, \tag{3.2}$$

where ϵ_t is an error term, and both ϕ_i and θ_i are coefficients to be estimated from the data. Regarding the TRAMO (Time Series Regression with ARIMA Noise, Missing Observations, and Outliers) method, it is responsible for identifying outliers and calculating regression variables, such as trading and moving holiday variables. SEATS (Signal Extraction in ARIMA Time Series) is then responsible for estimating the trend-cycle and seasonally adjusted component [44]. It is important to note that SEATS can introduce seasonality into the seasonal adjustment of a nonseasonal series [45]. Moreover, SEATS only works with quarterly and monthly data. Thus, other types of seasonality require different approaches [39].

Cleveland et al. [46] proposed a seasonal-trend decomposition procedure based on Loess (STL). This method is a filtering procedure for decomposing a time series into trend, seasonal, and remainder components. STL handles all types of seasonality and allows the specification of the seasonal and trend smoothing value. Moreover, the seasonal component is allowed to change over time, and the user can also control the change rate. Finally, STL is also able to decompose time series with missing values [39, 46]. It is also worth mentioning that we can use alternative methods that, instead of decomposing the time series into the aforementioned classical components, decompose it into wavelets [47], shapelets [48], and spectral components [47].

3.2 Time Series Data Analysis

This section focuses on the analysis of the relationship between time series, which can help describe how sensors are related. We can quantify the degree of the relationship by using a variety of measures, namely, similarity and correlation measures.

3.2.1 Similarity Measures

To determine how similar or different a pair of time series are, several measures have been proposed. The Euclidean distance, for example, is one of the most popular time series dissimilarity measures. This distance stands out due to its computational simplicity and indexing capabilities [49]. The Minkowsi

distance, also refereed to as ℓ_p-norm,

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{t=1}^{N} |x_t - y_t|^p \right)^{\frac{1}{p}}, \tag{3.3}$$

between two univariate time series, X and Y, of length N, can be considered a generalization of the Euclidean distance ($p = 2$) [50]. Other specific cases of the Minkowski distance are Manhattan ($p = 1$) and Chebyshev ($p \to \infty$).

Dynamic time warping (DTW) is an additional well-known similarity measure that focuses on finding the optimal alignment between two given, time-dependent, sequences. The sequences are warped in a non-linear fashion to match each other, minimizing the effects of shifting and distortion in time [51, 52]. For example, considering two univariate time series, X and Y, of length m and n, respectively, DTW creates an $N \times M$ matrix where its (i^{th}, j^{th}) element corresponds to the distance $d(x_i, y_j)$ [53]. Then, a warping path W uses a set of those elements to define a mapping between X and Y. To find the optimal warping path, i.e., the best alignment possible using DTW, we need to find the W that minimizes the warping cost,

$$\text{DTW}(X, Y) = \min \left(\sqrt{\sum_{k=1}^{K} w_k} \right), \tag{3.4}$$

where the k^{th} element of W is defined as $w_k = (i, j)_k$. It is also important to note that not only can DTW lead to counterintuitive alignments, but it can also fail to find obvious and natural ones. To address both problems, Keogh [54] proposed an extension of DTW called derivative dynamic time warping (DDTW) that considers the first derivative of the time series, i.e., their shape. Even though both DTW and DDTW have a running time of $O(MN)$, the computation time can be further improved through different techniques [52]. Piecewise dynamic time warping (PDTW) is also a method that intends to make DTW more efficient and faster [55]. This technique uses piecewise aggregate approximation that consists of compressing the time series into equally sized frames and calculating its mean value. Since the time series has now fewer time points, PDTW proves itself faster than DTW without decreasing the accuracy. Finally, Keogh et al. [53] later developed an even faster technique that uses a warping window width and a lower bounding function that discards sequences that could not possibly lead to the best alignment.

Another technique for similarity analysis is the Edit distance on real sequences (EDR) [49]. This method extends the original Levensthein distance [56] to real-valued time series. Chen et al. [57] reported that EDR outperformed previous Levensthein extensions for time series.

Marteau [58] proposed a particularly interesting similarity measure that essentially combines both DTW and EDR, called time-warped edit distance (TWED). This method takes time stamps differences into account, thus, being able to cope with time series of different sampling rates.

According to Serra and Arcos [49], TWED outperformed all distances considered in their study, including DTW, EDR, and the Euclidean distance. Additionally, DTW and EDR obtained similar results. Regarding the Euclidean distance, although it proved itself competitive, it generally performs statistically worse than TWED, DTW, and EDR.

3.2.2 Correlation Measures

To understand the degree of correlation between a pair of time series, we can consider classical and modern approaches. For example, Pearson's cross-correlation coefficient (PCC),

$$PCC(X,Y) = \frac{\sum_{t=1}^{M}(x_t - \bar{x})(y_t - \bar{y})}{\sqrt{\sum_{t=1}^{M}(x_t - \bar{x})^2} \cdot \sqrt{\sum_{t=1}^{M}(y_t - \bar{y})^2}}, \tag{3.5}$$

is one of the best-known correlation measures, where X and Y are two univariate time series of length M, and $\bar{x}$ and $\bar{y}$ are their corresponding means [59]. Instead of measuring the difference between time series values, PCC focuses on the relationship between time series. Thus, amplitude scaling or translation will not affect its result [59]. However, when the data has a specific distribution, such as non-linear distribution, PCC fails to detect the dependency between two or more variables [60]. It is also important to note that the sample size has a drastic effect on the results. De Winter et al [61] showed that in their dataset, a sample size of 25 was not enough to capture the true correlations. However, when doubling that size, the results got substantially better.

Rank correlation coefficients are used to measure an ordinal association, i.e., the extent to which one variable increases when the other increases without requiring that increase to be represented by a linear relationship. Both Spearman's and Kendall's rank correlation coefficients fall into this category. Since they are less sensitive to non-normality in distributions, they can be considered as an alternative to PCC [62].

Podobnik and Stanly [63] proposed a particularly interesting solution called detrended cross-correlation analysis (DCCA). This technique is based on detrended fluctuation analysis (DFA) and allows the analysis of two non-stationary time series. Considering X and Y as two univariate time series with M observations, each one being measured at a discrete time $t \in \{1, \ldots, T\}$, DCCA starts by defining $R_k \equiv \sum_{t=1}^{k} x_t$ and $R'_k \equiv \sum_{t=1}^{k} y_t$, where $k \leq T$. Then, it divides both time series into $T - n$ overlapping boxes, each containing $n + 1$ values, where $1 \leq n < T$. Considering that each box starts at t and ends at $t + n$, DCCA defines the local trend as $\tilde{R}_{k,t}$ and $\tilde{R}'_{k,t}$, where $(t \leq k \leq t + n)$. We now calculate, for each box, the covariance of its residuals,

$$f^2_{\mathrm{DCCA}}(n,t) \equiv (n-1)^{-1} \sum_{k=t}^{t+n} \left(R_k - \tilde{R}_{k,t}\right)\left(R'_k - \tilde{R}'_{k,t}\right). \tag{3.6}$$

Finally, to obtain the detrended covariance F^2_{DCCA}, we average the results of all boxes,

$$F^2_{\mathrm{DCCA}}(n) \equiv (T-n)^{-1} \sum_{t=1}^{T-n} f^2_{\mathrm{DCCA}}(n,t). \tag{3.7}$$

Since real-data can be categorized by a high degree of non-stationary, we can apply DCCA to a variety of fields, such as finance. However, when used in time series that exhibit periodicity, this method proves itself to be unreliable. Horvatic et al. [64] proposed a variation of DCCA by employing detrending with a varying polynomial order l. This technique (DCCA-$l(n)$) allows the analysis of non-stationary time

series with periodic trends. It is also worth noting that these methods do not quantify the level of cross-correlation. Nevertheless, to address this problem, Zebende [65] proposed a new and robust coefficient based on DFA and DCCA methods, that succeeds in identifying seasonal components in both types of cross-correlation (positive and negative).

3.3 Feature Extraction

Feature extraction consists of creating a new set of data by building new features from input data and then identifying the most relevant and informative features [66]. Accordingly, feature extraction has two optional steps, namely, feature construction and feature selection. It is also important to note that these steps can cause the loss of relevant information when not performed carefully. Note that temporal, spacial and geometric features can be extracted from time series using sliding time windows.

3.3.1 Feature Construction

Feature construction refers to transforming the original data into a set of useful features [66]. This transformation highly depends on the dataset and it might alter its dimensionality. Standardization and normalization are two methods that do not change it. Contrarily, principal component analysis (PCA) can be used to project the data into a lower-dimensional space by selecting the first principal components that comprise the highest original variance of the data, which can be useful for data visualization [67]. As an alternative to PCA, uniform manifold approximation and projection for dimension reduction (UMAP) is a nonlinear dimensionality-reduction technique already used in a variety of fields, including machine learning [68]. When dealing with complex problems, increasing the dimensionality space might also help.

3.3.2 Feature Selection

Feature selection methods are responsible for finding key features, i.e., relevant and informative features that represent our data. Additionally, it can increase an algorithm's speed and improve its performance [69]. One approach is through individual relevance ranking methods, which assume feature independence. Although these methods are considered fast and effective, they might penalize features that are individually irrelevant but become relevant when with others. Other approaches include forward and backward procedures that consist of adding or removing features based on a defined criterion [66].

Regarding individual relevance ranking methods, filter functions are the ones responsible for ordering the features according to their degree of dependence on the target [69]. It is important to note that filters are independent of the classifier's performance. Additionally, classical statistic tests, such as the Chi-squared, the Analysis of Variance (ANOVA) F-test, and the Kruskal–Wallis H test can be used as filter functions [66].

The Chi-squared test is used when in the presence of discrete features and measures the difference between the observed counts $o_{\mathbf{i}}$ and the expected counts $e_{\mathbf{i}}$ of the data [10]. Therefore, it is defined as

$$\chi^2 = \sum_i \frac{(o_i - e_\mathbf{i})^2}{e_i}. \tag{3.8}$$

For continuous features, we can use the ANOVA F-test instead. It computes the ratio between the mean square error between groups MS_{bn} and the mean square error within groups MS_{wn} as in

$$F = \frac{MS_{\text{bn}}}{MS_{\text{wn}}}. \tag{3.9}$$

The mean square error between groups,

$$MS_{\text{bn}} = \sum_{i=1}^{K} n_i \left(\bar{X}_i - \bar{X}\right)^2 /(K-1), \tag{3.10}$$

is an estimate of the differences between means found in the data, where n_i denotes the number of instances in the i^{th} group, $\bar{X}_i$ represents the sample in the i^{th} group, $\bar{X}$ is the overall mean of the data, and K is the number of groups. Finally, the mean square error within groups,

$$MS_{\text{wn}} = \sum_{i=1}^{K} \sum_{j=1}^{n_i} \left(X_{ij} - \bar{X}_i\right)^2 /(N-K), \tag{3.11}$$

is an estimate of the variability of individual scores in the data, where X_{ij} is the j^{th} instance in the i^{th} group, and N is the number of instances in the data [70]. However, the ANOVA F-test requires the features to follow a normal distribution.

For non-normal distributions, we can use the Kruskal–Wallis H test. Since it is a non-parametric test, it assumes no specific distribution [71]. This test starts by assigning ranks to the data using all scores in the experiment. Then, it computes the sum of squares between groups,

$$SS_{\text{bn}} = \frac{(\Sigma R_1)^2}{n_1} + \frac{(\Sigma R_2)^2}{n_2} + \cdots + \frac{(\Sigma R_k)^2}{n_k}, \tag{3.12}$$

where K is the number of groups, n_i denotes the number of instances in the i^{th} group, and ΣR is the sum of the ranks in each group. Lastly, it computes the value of H as

$$H = \left(\frac{12}{N(N+1)}\right)(SS_{\text{bn}}) - 3(N+1), \tag{3.13}$$

where N is the number of instances in the data [70]. If H exceeds a critical value, we may conclude that the groups do not come from the same population [71].

3.4 Burst Leakage Detection

Since the main focus of this work is burst leakage detection, we explored different methods that detect and discover leakages. These methods can be grouped into hardware-based methods, which aim to accurately locate the leakage, and software-based methods, which attempt to isolate possible leakage areas [72].

3.4.1 Hardware-Based Methods

Hardware-based leakage detection methods can be grouped into two categories, namely, leakage localization methods, and leakage pinpointing methods [73]. Leakage localization methods, such as acoustic logging and ground-penetrating radars, help with the identification and prioritization of leakage areas to later facilitate the discovery of the exact leakage location. Finally, leakage pinpointing methods, such as leakage noise correlators and gas injection, are one of the most precise technologies currently available. Although the accuracy of these methods is high, they are slow to run, the equipment and man-hours are expensive, and may also require the interruption of the network [72, 73]. Consequently, it led to a search for faster and cheaper techniques, such as the ones based on software.

3.4.2 Software-Based Methods

Software-based leakage detection methods usually rely on a model or algorithm that uses additional information, such as pressure or volumetric flowrate data, rather than leakage noise information, and can be grouped into numerical and non-numerical methods [72]. The numerical ones mainly include methods based on transient events and on the traditional hydraulic model method, whereas the non-numerical ones use a data-driven approach. These methods analyze the monitoring data and find their rules by using data mining and artificial intelligence algorithms. It is important to note that some methods, for example, methods that use artificial neural networks, need to be updated frequently and require a lot of historical data to be trained [74].

Valizadeh et al. [75] proposed a solution using feature extraction and classification. They used time windows and time-domain features to transform the time signals of flow, pressure, and temperature at the inlet and outlet of the network into a matrix of features. The time-domain features include the mean value, root-mean-square, standard deviation, among others. Regarding the classifiers, they compare a k-Nearest Neighbors classifier to a Bayesian classifier. Although it is not clear which one is better, these two classifiers perform differently depending on the window size and leak characteristics, i.e., its diameter and location.

3.5 Other Applications

In addition to burst leakage detection, the contributions proposed along the thesis have applicability for three additional tasks. The first one is related to background leakages, which can go undetected for long periods of time [20]. So, even with a small water loss rate, staying unrepaired for a long time can still have a substantial impact on the amount of water wasted. The second topic is related to sensor placement since insufficient sensor coverage has a direct negative impact on the identification and localization of network events like leakages. The last topic is related to the assessment of the status of active hydraulic elements, such as valves. The operation of these elements can results in the isolation of certain parts of the network, altering its structure. Therefore, it would be interesting to have a system that automatically detects these events and determines the exact network structure at a given moment.

3.5.1 Background Leakage Detection

Upon further research, we identified that most techniques focus on detecting burst leakages instead of background leakages. Unlike burst leakages, the background type is not characterized by a sudden pressure drop, thus, making techniques based on pressure sensor data ineffective. Moreover, since background leakages increase with pipes' internal pressure, we can minimize water losses by decreasing the excessive pressure at strategic points. Therefore, Adedeji et al. [76] proposed an algorithm that not only identifies critical pipes given a leakage, i.e., pipes experiencing large background leakage outflows, but also indicates the probable pipes of the network where pressure control should be performed. However, this technique does not identify the exact point on the network where the background leakage occurs.

3.5.2 Sensor Placement

We found two main categories of sensor placement strategies for WDNs, namely, the placement of water quality sensors for contamination monitoring, and the placement of flow and pressure sensors for leakage localization [77, 78]. Since this thesis centers around flow and pressure data, we focused on the works that fall into the second category.

Sensor placement can be seen as an optimization problem since we need to identify which is the best distribution of a limited number of sensors. This will allow a more effective detection and localization of network events, such as leakages and closing of valves. Pérez et al. [79] developed a sensor placement method for Barcelona's WDN that uses a pressure sensitivity matrix. This matrix allows the representation of how sensitive each sensor is to each possible leakage. However, due to data availability, this matrix can be extremely difficult to calculate for a real network, thus, enforcing the authors to use a simulation. Regarding the sensor placement itself, this technique uses a genetic algorithm that converges to an optimal combination, i.e., the combination of sensors that are more sensitive to all the possible leakages. Finally, this method presents good results under ideal conditions. However, its performance decreases in the presence of nodal demand uncertainty and noise in the measurements [77].

Serrate et al. [80] proposed a solution based on a sensitivity matrix similar to the one used in Pérez et al. [79]. However, instead of using genetic algorithms to find the optimal sensor combination, this technique uses clustering, more specifically, the k-means algorithm. It consists of grouping sensors based on their sensitivity and then selecting a representative for each cluster. The chosen representatives are the final sensors used in the WDN. Since we do not need to explore every single sensor combination, getting the result through clustering techniques is must faster. Nonetheless, k-means does not guarantee a global optimal solution, meaning that other clustering algorithms should be considered.

3.5.3 Active Hydraulic Elements Assessment

Assessing the status of active hydraulic elements, such as valves, was not addressed in most proposals. However, we were able to find one by Stephens et al. [81] that covered the problem. The authors use inverse transient analysis to determine if a valve is closed or open. More specifically, they use the transient model of the network to determine the predicted response for different combinations of valve status. Then, they

use a genetic algorithm to determine which combination suits the measured responses. It is also worth mentioning that the authors carefully used optimization strategies to minimize the number of possible combinations, thus reducing the algorithm's running time. Nevertheless, this proposal does not consider an uncertain environment, such as the presence of leakages, possibly compromising its application to a real network. Furthermore, this proposal does not solve our problem entirely, since instead of dynamically assessing the status of a single valve, this technique passively determines the status of all valves present in the network.

Regarding other proposals that we found in this area, although not directly related, they are still worth noting. For example, most proposals focus on modeling combinations of active hydraulic elements, thus, allowing WDN models to become more similar to real networks [82, 83]. Other proposals study the impact of a given network interruption, allowing the identification of consequently disconnected pipes from the WDN [84]. Additionally, another proposal finds the best valve configuration to isolate a given section of the network [85]. Therefore, this technique is capable of minimizing the impact of interventions on the network. So, although most of the explored proposals do not answer our problem directly, they can still be useful for managing both planned and unplanned interruptions of the supply of water.

Chapter 4

Solution

The goal of this work is to detect burst leakages in water distribution networks through the classification of time series data collected by their sensors. Accordingly, we divide our solution into the three steps described in Figure 4.1. Section 4.1 presents the original datasets, the problems we found, and how

Figure 4.1: Solution pipeline.

we tackled along the preprocessing stage. Section 4.2 proposes a new class of descriptive models of network dynamics through the correlation analysis of pairs of sensors, i.e., describes how we transformed our original dataset into a set of objective indicators. Lastly, section 4.3 proposes predictors of leakage dynamics using the descriptors found and further describes the hyperparameter tuning and performance assessment.

4.1 Dataset Description and Preprocessing

As the study case WDN, Infraquinta's network was selected. Infraquinta manages Quinta do Lago, a tourist resort with extensive irrigation, large hotel units, and irregularities in the occupation of households. Therefore, its WDN suffers from highly irregular consumption patterns, which will allow us to test our solution in a challenging setting. The network is equipped with precise sensors, leading to more accurate time series. Its sensors also cover a larger area and are closer to the delivery points, allowing a better understanding of the consumption patterns. However, for the prediction task, we only used data from telemanagement sensors since the telemetry sensors are less prone to be affected by leakages due to their proximity to the delivery points. We also have an EPANET model of Infraquinta's network at our disposal, which allows the generation of data in a controlled environment, i.e., without noise and network changes, which we used to conduct the first experiments. Accordingly, sections 4.1.1 and 4.1.2 describe the characteristics and problems of the synthetic data and the real data, respectively.

4.1.1 Artificial Water Distribution Network

The synthetic data was generated using an EPANET model of Infraquinta's WDN, originating 18696 chunks of volumetric flowrate and pressure data. The flowrate data comprises data extracted from 7 network points, which are equivalent to the location of the flowrate sensors. According to WU experts, the location of the pressure sensors was not suitable to identify all leakages. So, instead, they decided to extract the pressure data from 21 relevant network points, where new pressure sensors could be added. So, with the 7 flowrate and the 21 pressure network points, we have data from a total of 28 points. For simplification purposes, we will refer to these network points as sensors.

Each chunk has 145 time points and a granularity of 600 seconds, i.e., each chunk equals a day of data. Furthermore, each one of these chunks includes a leakage with a different combination of location and size. The leakages were generated using six leakage coefficients, namely, 0.05, 0.1, 0.5, 1.0, 1.5 and 2.0. The higher the coefficient, the larger the leakage is. In that way, our data covers all 3116 points of the network and six leakage sizes ($18696 = 3116 \times 6$). These leakages run for 24 time points (approximately 4 hours) and can happen anywhere in the chunk.

Regarding the preprocessing, we found two minor problems with the synthetic data. The first one is related to negative measurements found in the data, which were caused by the configuration of pipe directions in the EPANET model. The absolute value of the measurements is therefore considered. Moreover, we found measurements very close to zero in flowrate sensors. Since there was no evidence of the existence of water flow in those situations, we approximated to zero all flowrate values bellow 1×10^{-3}.

4.1.2 Real Water Distribution Network

The available real data corresponds to the entire year of 2017 and was extracted from 7 volumetric flowrate and 6 pressure sensors. Unlike synthetic sensors, have informative designations. For simplification purposes, we will address these sensors by their database identifiers (IDs) instead, as shown in Table 4.1. The WDN monitoring system has a granularity of approximately 60 seconds and is missing the

ID	Volumetric Flowrate Sensors
1	APA Caudal Atual
2	PB2 Caudal Caixa 1
6	RSV R5 Caudal Caixa
9	QV Caudal
10	HC Caudal
12	RPR Pre
14	RPR Caudal Grv

ID	Pressure Sensors
3	PB2 Pressão Caixa 1
7	RSV R5 Pressão Caixa 2
8	QV Sonda de Pressão
11	RPR Pressão Pre
13	RPR Pressão Grv
15	APA Pressão

Table 4.1: Correspondence between the real sensors' names and ids.

last day of every month. Regarding leakages, out of 16 occurrences reported in 2017, we have complete information about 12, as shown in Table 4.2. It is also important to note that although we know the time leakages were reported, we do not know when they actually started nor its size.

Concerning the preprocessing, since we only have 12 days missing out of 365, we did not apply any data

	Reported Time	Beginning of Resolution	End of Resolution
1	2017/01/08 08:30	2017/01/08 09:00	2017/01/08 14:00
2	2017/02/07 12:10	2017/02/07 12:15	2017/02/07 16:00
3	2017/05/01 04:20	2017/05/01 04:45	2017/05/01 10:35
4	2017/05/07 08:25	2017/05/07 09:15	2017/05/07 17:30
5	2017/05/12 11:15	2017/05/12 14:00	2017/05/12 16:20
6	2017/06/13 10:18	2017/06/13 11:03	2017/06/13 14:47
7	2017/07/05 03:00	2017/07/05 03:30	2017/07/05 10:45
8	2017/09/09 09:00	2017/09/09 09:15	2017/09/09 12:30
9	2017/09/12 09:35	2017/09/12 09:40	2017/09/12 11:30
10	2017/11/01 19:02	2017/11/01 19:30	2017/11/02 10:57
11	2017/11/08 16:40	2017/11/08 17:30	2017/11/08 20:30
12	2017/12/12 12:26	2017/12/12 13:40	2017/12/12 16:50

Table 4.2: Reported leakages in 2017.

imputation technique and used the remaining days instead. The other problem found is irregular time steps of recording. To estimate observations at a regular sampling (equally distant points), we applied the linear interpolation method described in section 3.1.1 and available in Pandas[1] Python module.

4.2 Descriptors of Dynamics: Correlation Analysis

In this section, we explain how we transformed our original time series dataset into a new dataset without explicit feature ordering and how we selected the most informative features. It is also important to remember that the disruption of the correlation between two sensors may indicate a leakage. The use of proper correlations captures different forms of relationships:

- Sensors of different types since pressure and volumetric flowrate sensors are inversely correlated;
- Sensors from pipes with distinct characteristics, such as diameter and slope;
- Sensors with various consumption patterns, i.e., flowrate differences explained by additive factors.

4.2.1 Feature Construction

The correlation value between two highly correlated sensors is expected to decrease when a leakage occurs between them. In light of this, we created a feature construction strategy that consists of calculating the correlation value between all pairs of sensors throughout time. Regarding the correlation methods, we used two correlation methods described in section 3.2.2. The first one is the classical and widely used Pearson cross-correlation coefficient (PCC), available in SciPy[2] Python module. Since our time series suffer from irregularities, we also decided to use the modern approach cross-correlation analysis (DCCA), available in Jaime Ide's profile on GitHub[3].

[1] `https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.interpolate.html`
[2] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.pearsonr.html`
[3] `https://gist.github.com/jaimeide/a9cba18192ee904307298bd110c28b14`

Before we explain our strategy, note that all instances of the new dataset have a class. In our case, the positive class corresponds to a change in the elements' state, namely the presence of a leakage, while the negative class corresponds to remaining in the same state. Concerning our strategy, each column of our dataset shows the correlation between a pair of sensors and each row indicates the time window in which those values occurred.

Regarding the synthetic dataset, each chunk originated two instances, specifically, one positive and one negative. To create the positive instances, either the beginning or the end of the leakage is in the middle of the time window, as Figure 4.2 shows. Then, to create the negative instances, we placed the

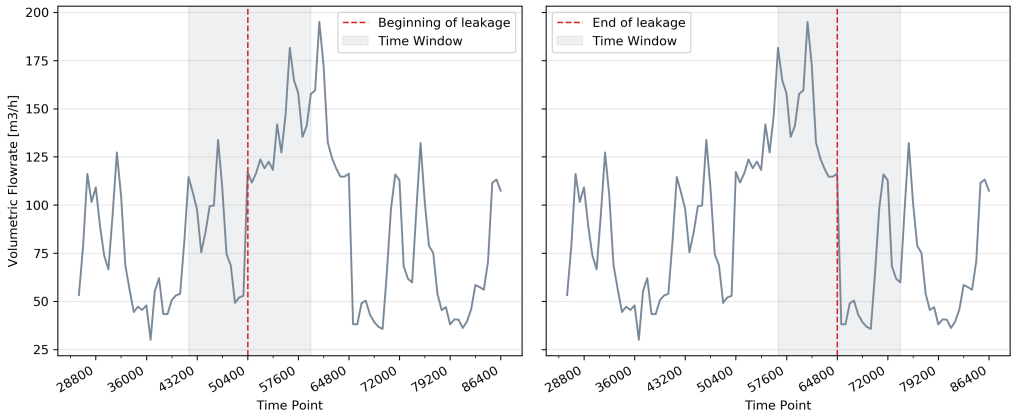

Figure 4.2: Time window configurations for the positive class of the synthetic data.

time window either before or after the leakage, as Figure 4.3 depicts. So, each instance of the dataset

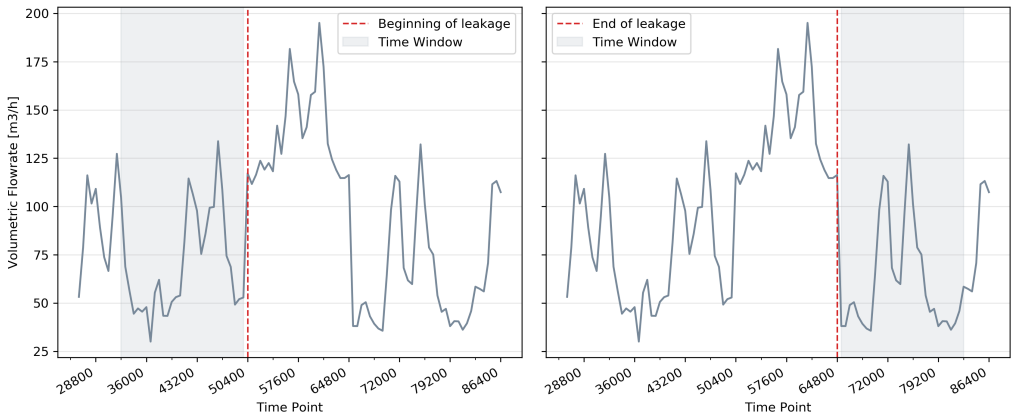

Figure 4.3: Time window configurations for the negative class of the synthetic data.

comprises the correlation values between all pairs of sensors during a time window. Regarding the time window size, we generated different datasets with time windows between 16 (2h40) and 40 (6h40) time points.

For the real dataset, we have a sliding window that moves over the time series in intervals of 15 minutes, allowing the early detection of leakages and avoiding missing their beginning. We considered as positive all instances that occurred between the leakage report and the beginning of the resolution work. Since we do not know the exact time the leakage started, we also counted as positive the instances that

took place two hours before the official leakage report time. It is also important to note that the instances that occurred during the resolution of the leakage are not part of the leakage nor the everyday behavior of the network. Therefore, we did not consider them as part of the new real dataset. Finally, regarding the time window size, we generated different datasets with time windows between 60 and 240 minutes. Figure 4.3 shows an example of a real dataset with three sensors using a time window of 60 minutes.

Time Window	Flowrate Sensor 1 Flowrate Sensor 2	Flowrate Sensor 1 Pressure Sensor 3	Flowrate Sensor 2 Pressure Sensor 3	Class
2017/01/08 05:15 - 2017/01/08 06:15	DCCA/PCC	DCCA/PCC	DCCA/PCC	Negative
2017/01/08 05:30 - 2017/01/08 06:30	DCCA/PCC	DCCA/PCC	DCCA/PCC	Negative
2017/01/08 05:45 - 2017/01/08 06:45	DCCA/PCC	DCCA/PCC	DCCA/PCC	Positive
2017/01/08 06:00 - 2017/01/08 07:00	DCCA/PCC	DCCA/PCC	DCCA/PCC	Positive

Table 4.3: Example of a real dataset with three sensors and a time window of 60 minutes.

4.2.2 Feature Selection

Since not all sensors show a significant amount of correlation with each other, we might not need all features generated in the construction step. Additionally, reducing the number of features might (1) lead to fewer sensors needed in the network and (2) improve the classifiers' speed and performance. Regarding the feature selection methods, since our data does not follow a normal distribution, we used the non-parametric Kruskal–Wallis H test available in SciPy[4] Python module and explained in section 3.3.2. Additionally, we also created our own feature ranking approach that favors strong correlations in normal situations, i.e., without the presence of a leakage.

In our feature ranking approach, we start by (1) finding the mean of each pair (feature) within negative instances. Then, (2) for each sensor, we sort its pairs by mean. Lastly, (3) we take the pairs that ranked first/higher for each sensor, remove the duplicates, and sort them. We do that for the ones that ranked second and so on. As we can see, this strategy prefers the highest correlated pairs without favoring one sensor in particular.

4.3 Predictors of Leakage Dynamics: Burst Detection

For the prediction of burst leakages, we chose the three classifiers described in 2.1.3, namely, the NB classifier available in Scikit-Learn[5] Python module, an SVM with a linear kernel also available in Scikt-Learn[6], and an SVM with an RBF kernel available in Scikit-Learn[7] as well. The remainder of this section explains the strategy used to assess the classifiers' performance and tune their hyperparameters.

[4] https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mstats.kruskalwallis.html
[5] https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
[6] https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html
[7] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

4.3.1 Performance Assessment

Here we introduce the key concepts of performance assessment and present the performance measures used to support the tuning of the hyperparameters of our classifiers.

Remember that the positive class identifies events of interest, which in our case corresponds to the existence of a leakage. Accordingly, each classified instance falls into one of the four categories:

- *True Positives (TP):* Positive instances classified as positive.
- *False Negatives (FN):* Positive instances classified as negative.
- *True Negatives (TN):* Negative instances classified as negative.
- *False Positives (FP):* Negative instances classified as positive.

A quick way to visualize these four categories, i.e., the classification results, is through a 2×2 confusion matrix, as shown in Table 4.4. Each row of the matrix represents the number of instances in the true class, whereas each column represents the number of instances in the predicted class [10].

	Predicted Class	
True Class	Negative (Non-Leakage)	Positive (Leakage)
Negative (Non-Leakage)	TN	FP
Positive (Leakage)	FN	TP

Table 4.4: Confusion matrix [10].

To assess the model's performance, we can use global measures and class-specific measures depending on our needs [10]. The accuracy,

$$Accuracy = \frac{TP + TN}{TP + FN + TN + FP}, \tag{4.1}$$

is a global measure and corresponds to the fraction of the correct predictions, giving an overall idea of the model's performance. However, when in the presence of an imbalanced dataset, the accuracy camouflages the results of the underrepresented class. Therefore, we used class-specific measures, namely, the recall, the precision, and the F-measure. The recall for the positive class, also referred to as true positive rate (TPR),

$$Recall_P = TPR = \frac{TP}{TP + FN}, \tag{4.2}$$

represents the portion of the positive instances correctly identified, while the precision for the positive class,

$$Precision_P = \frac{TP}{TP + FP}, \tag{4.3}$$

shows the portion of the positive instances correctly identified among all instances classified as positive. Lastly, the F-measure for the positive class,

$$F_P = \frac{2 \times precision_P \times recall_P}{precision_P + recall_P}, \tag{4.4}$$

corresponds to the harmonic mean of the precision and recall, and its highest possible value is one.

4.3.2 Threshold Adjustment

A binary classifier usually chooses a positive score threshold ρ to determine if the instances are positive or negative. In particular, the model classifies all instances above ρ as positive, and the remaining ones as negatives. However, the default threshold might be preventing the model from achieving optimal results. In other words, it is essential to minimize the number of FN (unidentified leakages) and the default threshold might be minimizing FP instead. To identify the optimal threshold, receiver operating characteristics (ROC) analysis will help us with that.

The ROC analysis is responsible for analyzing the performance of the model over all possible values of the threshold [10]. For each value of ρ, it plots the false positive rate (FPR),

$$FPR = 1 - Recall_N = 1 - \frac{TN}{TN + FP}, \tag{4.5}$$

on the x-axis, and (4.2) on the y-axis, resulting in a curve similar to the one shown in Figure 4.4. As we can see, the ROC curve represents the trade-off between the TPR (rising as we move up) and the FPR (dropping as we move right). Since the best-case scenario occurs when the model correctly identifies all

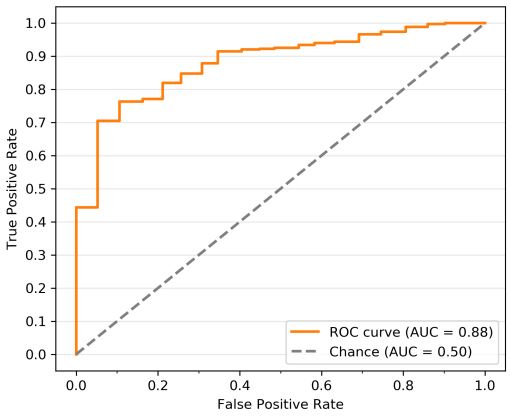

Figure 4.4: Example of a ROC plot.

instances, i.e., $TPR = 1$ and $FPR = 0$, the threshold we should choose corresponds to the closest point to the upper left corner. When in the presence of more than one possible threshold, we should choose the one that minimizes FN, that is, the number of unidentified leakages. It is also important to note that the final threshold of the models was picked through a method called cross-validation explained in section 4.3.3.

Regarding the evaluation of the model, it is also important to mention the area under the ROC curve (AUC). Since the model's performance is better when its ROC curve is closer to the upper left corner, the ideal value of AUC is the total area of the ROC plot, namely $AUC = 1$. Note that the diagonal dashed line, $AUC = 0.5$, represents the ROC curve of a random classifier. So, if a model has an AUC value bellow that, its performance is worse than random [10].

Although all thresholds ρ determine an instance's class, its meaning and values differ from classifier

to classifier. In SVMs' case, their default threshold corresponds to the separating hyperplane $h(x) = 0$, i.e., $\rho = 0$. Therefore, the threshold would have to be negative to minimize FN in a non-separable case.

NB's situation is more complex since it does not have a threshold originally. Instead, it calculates the probability of an instance being positive or negative and chooses the class with the highest score. However, to minimize the FN, we need to guarantee that NB only chooses the negative class when it is certain, i.e., if the difference to the positive class is substantial. Accordingly, to define a threshold, the score NB calculates for an instance needs to correspond to the difference between the probability of it being positive and the probability of it being negative, resulting in $-1 \leq \rho \leq 1$. For example, a threshold of -0.7 means that the instance will only be classified as negative if the difference to the probability of being positive is higher than 0.7. On the other hand, a threshold of 0.7 means that the instance will only be classified as positive if the difference to the probability of being negative is higher than 0.7. Note that if we set our threshold to zero, NB returns to its normal behavior.

4.3.3 Evaluation Strategy

The simplest evaluation strategy would be to randomly split the dataset into a training set, a validation set, and a test set. We use the training set to train the model and then the validation set to tune its parameters and guarantee their generalization ability. When the experiments seem successful, we use the test set to perform the final evaluation. However, two problems arise, namely, (1) splitting the dataset into three sets reduces the available instances to train the model and (2) the algorithm's performance depends on the split, resulting in an unreliable evaluation [10]. To minimize these drawbacks, a K-fold cross-validation strategy was applied.

The K-fold cross-validation strategy, as Figure 4.5 shows, consists in splitting the data instances into K equal-size parts, called folds. Each fold takes a turn into being the validation set, while the others act as the training set. If we get satisfying and consistent results through all iterations, it is a good indication that the model is not overfitting the data, i.e., the model is generic enough to accommodate small changes in the data. Then, we use the test set to conduct the final evaluation [10].

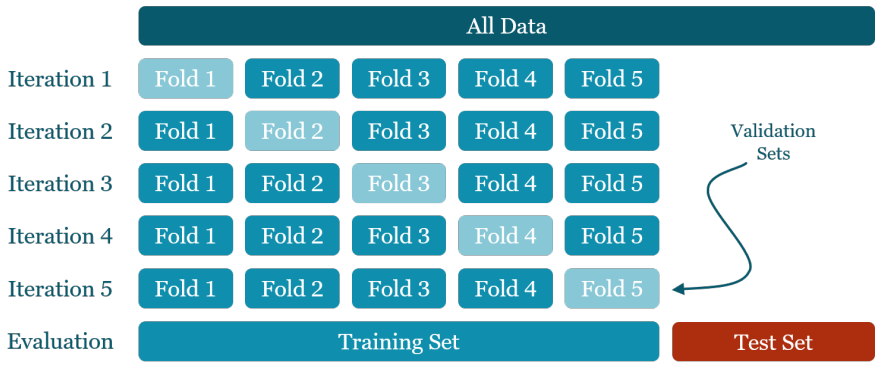

Figure 4.5: 5-Fold cross-validation example.

Regarding the synthetic data, we put aside 20% of our dataset to serve as the test set and used a 5-fold cross-validation on the remaining data. We shuffled the dataset before splitting just in case the instances in our dataset had a specific order. Moreover, since we have different types of leakages, we equally split

them between folds. We also guaranteed that the number of positive and negative instances is the same within folds. Concerning the real data, severe class imbalance is observed due to the scarce presence of 12 monitored leakages in total. In this context, the number of folds used is equal to the number of leakages, ensuring one leakage in the validation set, while the others are used for training. Please note that one leakage is equivalent to more than one positive instance due to the sliding time windows. We also put one of the folds aside to serve as the test set.

Chapter 5

Results: Descriptive Setting

This chapter presents a comprehensive analysis of the steps performed to learn the descriptive models of network dynamics. In section 5.1, we start by exploring our original datasets through visualization methods and the extraction of its main characteristics. Then, section 5.2 overviews the datasets created through feature extraction. Section 5.3 shows how the correlation changes over time, with particular regard to leakage occurrences. Then, section 5.4 assesses how the correlation methods respond to different leakage sizes. Section 5.5 presents how different time window sizes affect the correlation. Finally, section 5.6 focuses on understanding how DCCA parameterization affects the overall correlation.

5.1 Exploratory Data Analysis

Exploratory data analysis is the process of extracting the main characteristics of the dataset's features through statistics and visualization methods, allowing a better understanding of the dataset. With this in mind, this section focuses on analyzing the original datasets, describing the location of the sensors, how they relate to each other, and presenting a quantitative summary of the features.

5.1.1 Artificial Water Distribution Network

As previously mentioned, we have a total of 28 sensors placed throughout the artificial WDN. The 7 volumetric flowrate sensors follow the distribution of the real ones, i.e., are placed in three distinct areas, as Figure 5.2 shows. Concerning the 21 pressure sensors, we know that they are in relevant network points, but we do not know their exact location.

As expected, and shown in Figure 5.1, we can identify a clear seasonality in the synthetic time series. In these conditions, disruptions in the network are more noticeable, which is the ideal setting for the first experiments. We can also see that when the flowrate drops, the pressure increases, and vice-versa, which is a sign of inverse correlation.

Lastly, Tables 5.1 and 5.2 present the descriptive statistics of the synthetic dataset, i.e., a quantitative summary of the synthetic sensors from chunk 697, taken as an illustrative example. By observing the standard deviation entries in these two tables, we can see that the volumetric flowrate varies a lot more

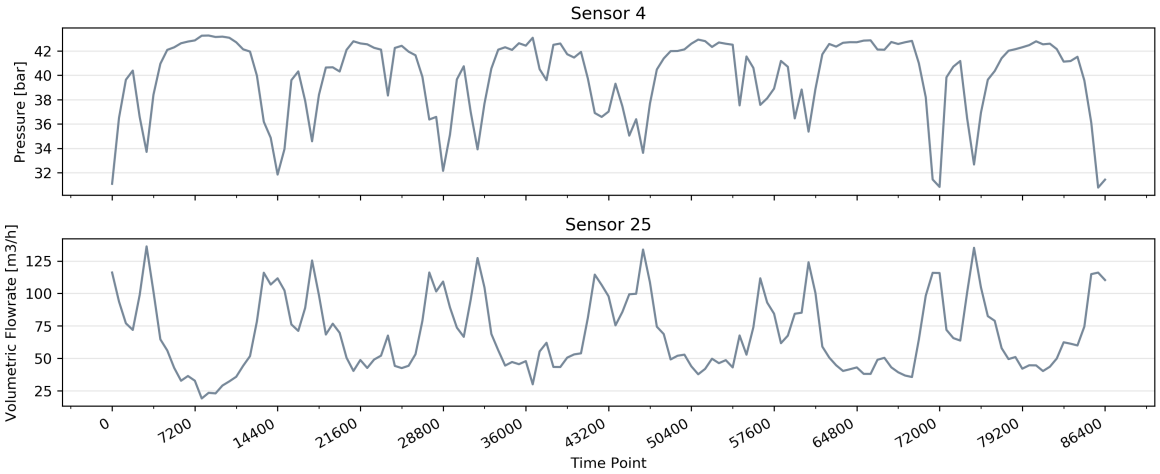

Figure 5.1: Synthetic time series of sensors 4 and 25 from chunk 697.

than the pressure. However, as Table 5.2 shows, sensors 27 and 28 are, respectively, always and mostly zero. Considering that this happens in most chunks, we will not be using them in the next steps.

	Pressure Sensors													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
mean	23.13	39.43	60.10	39.91	27.37	40.71	34.01	43.92	48.14	37.59	48.80	33.30	48.33	43.53
std	1.26	1.09	0.56	3.23	0.08	1.33	1.97	2.15	2.03	2.08	1.97	1.97	1.98	2.83
min	19.43	36.21	58.47	30.78	27.07	35.57	26.82	35.51	40.92	29.35	41.52	26.12	41.11	35.82
25%	22.50	38.87	59.67	37.90	27.35	40.30	33.03	43.01	47.00	36.57	47.84	47.84	32.33	47.36
50%	23.54	39.79	60.30	41.17	27.39	41.12	34.20	44.12	48.34	37.64	48.97	48.97	33.50	48.54
75%	24.13	40.29	60.55	42.44	27.43	41.57	35.42	45.51	49.69	39.15	50.22	50.22	34.71	49.74
max	24.73	40.82	60.78	43.27	27.47	42.40	38.06	48.11	52.25	41.69	52.85	37.36	52.41	46.58

Table 5.1: Descriptive statistics of the synthetic pressure sensors (1 to 14) from chunk 697.

	Pressure Sensors							Volumetric Flowrate Sensors						
	15	16	17	18	19	20	21	22	23	24	25	26	27	28
mean	42.76	30.66	43.80	32.56	34.80	25.94	25.99	8.63	87.68	177.68	68.68	28.96	0.00	0.24
std	4.66	4.59	4.53	0.61	0.49	0.77	0.45	5.50	18.30	35.93	28.46	11.69	0.00	1.32
min	30.84	19.30	32.66	30.66	33.14	23.28	24.48	1.66	40.77	97.86	19.15	9.00	0.00	0.00
25%	41.35	40.33	28.29	41.44	32.16	34.52	25.56	4.84	77.36	149.52	44.61	19.56	0.00	0.00
50%	44.66	44.04	31.98	45.09	32.59	34.85	26.20	7.06	89.02	177.39	62.03	26.43	0.00	0.00
75%	45.83	46.24	34.09	47.21	33.01	35.18	26.48	10.48	97.87	199.23	93.02	36.41	0.00	0.00
max	48.76	36.57	49.66	33.65	35.66	26.78	26.79	30.18	148.58	279.15	136.23	58.02	0.00	11.77

Table 5.2: Descriptive statistics of the synthetic pressure sensors (15 to 21) and volumetric flowrate sensors (22 to 28) from chunk 697.

5.1.2 Real Water Distribution Network

Unlike the artificial WDN, all 13 sensors from Infraquinta are concentrated in three areas, as Figure 5.2 shows. Moreover, nine of them are located in the same area, leaving one area with three sensors and the other with only one. Consequently, the lack of sensor coverage may negatively impact the ability to

Figure 5.2: Overview of the real sensors' location in Infraquinta's WDN.

detect leakages that do not occur between these areas.

As expected, the time series from Infraquinta's WDN are generally more complex than the synthetic ones, i.e., contain more noise, are vulnerable to network changes and are exposed to different consumption patterns. All these factors make the analysis and visualization of time series more difficult. Hence, to understand how our time series behave, we decomposed two weeks of data from sensor 6. Regarding the method used, we applied an additive decomposition model described in section 2.1.1 and available in Statsmodels[1] Python module. Although we were able to extract a clear seasonality, we can see in Figure 5.3 that the time series contains large residuals and its trend does not convey enough information about what happened during these two weeks. For example, considering the trend component, we see an increase of the flowrate from May 15 (Monday) through 19 (Friday), but that does not happen from May 22 (Monday) through 26 (Friday). Ideally, we should have smaller residuals and an informative trend.

Lastly, Table 5.3 presents the descriptive statistics of the real dataset. Through the analysis of

	Pressure Sensors						Volumetric Flowrate Sensors						
	3	7	8	11	13	15	1	2	6	9	10	12	14
mean	5.62	2.97	2.80	2.30	0.40	2.51	57.99	30.83	98.80	18.86	6.75	155.29	68.36
std	0.43	0.31	0.01	0.01	0.00	0.03	34.05	14.19	32.67	12.22	12.08	47.43	36.04
min	3.80	1.60	2.60	2.10	0.40	1.96	4.44	0.00	31.30	1.50	0.00	58.91	5.10
25%	5.40	2.80	2.80	2.30	0.40	2.50	28.94	20.70	75.59	9.20	0.00	119.21	38.30
50%	5.73	3.09	2.80	2.30	0.40	2.50	49.14	28.45	91.36	14.53	0.50	146.42	63.30
75%	5.93	3.20	2.80	2.30	0.40	2.50	83.70	39.41	119.80	27.20	7.10	186.86	94.30
max	6.30	3.50	3.03	2.40	0.40	2.70	159.80	78.46	208.30	56.20	51.40	331.95	186.37

Table 5.3: Descriptive statistics of the real sensors from May 15 through 28, 2017.

the standard deviation entries, we can see that the volumetric flowrate also varies a lot more than the

[1] `https://www.statsmodels.org/stable/generated/statsmodels.tsa.seasonal.seasonal_decompose.html`

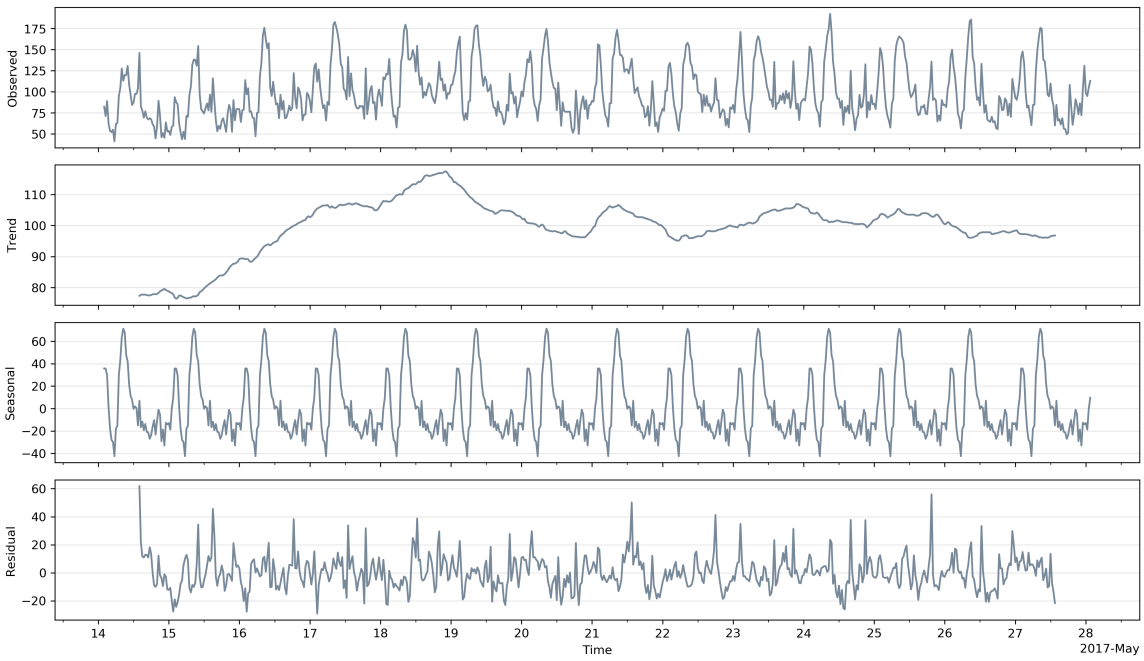

Figure 5.3: Additive decomposition of the real flowrate time series from sensor 6 from May 15 through 28, 2017.

pressure. Additionally, sensors 8, 11, 13, and 15 appear to be almost constant during the whole year, not only during those two weeks. Considering that this happens consistently throughout the year, we will not be using them in the next steps.

5.2 Correlation Analysis

This section describes the relationships between the WDN sensors in accordance with the proposed feature construction step. To get an overall idea of the values of each feature from paired sensors, heatmaps are used. Heatmaps are matrices where each cell encodes a single quantitative value with color [86]. In our case, that value corresponds to the correlation value (DCCA or PCC) between a pair of sensors. Therefore, one heatmap corresponds to one instance of our dataset. Regarding the colors of the heatmap, we used a double-ended hue scale. The blue scale represents a positive correlation, while the red scale represents an inverse (also referred to as negative) correlation.

Regarding the artificial WDN, since we considered 26 synthetic sensors, our new synthetic dataset has a total of ${}^{26}C_2 = 325$ features. Figure 5.4 represents the heatmaps of two synthetic instances, one positive and one negative, from the same chunk, using DCCA as the correlation method. Figure 5.5 also shows those two instances but, this time, using PCC as the correlation method. Focusing on the negative instances, we can see that pairs of sensors of the same type are directly correlated, while pairs of different types are inversely correlated. DCCA and PCC also obtained similar color patterns, but PCC looks more robust against network disruptions since its correlations are stronger in both instances. Consequentially, the difference to the negative instance with PCC is less noticeable than with DCCA. Therefore, DCCA may be more sensitive to leakages and contribute to better results in the classification step than PCC.

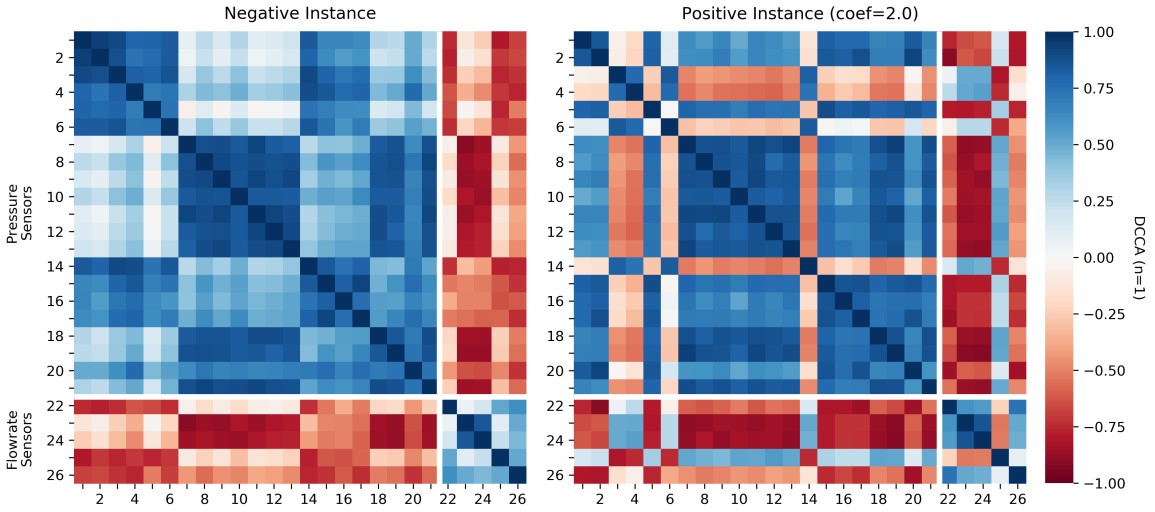

Figure 5.4: DCCA heatmaps of two synthetic instances from chunk 702 using a time window of 40 time points.

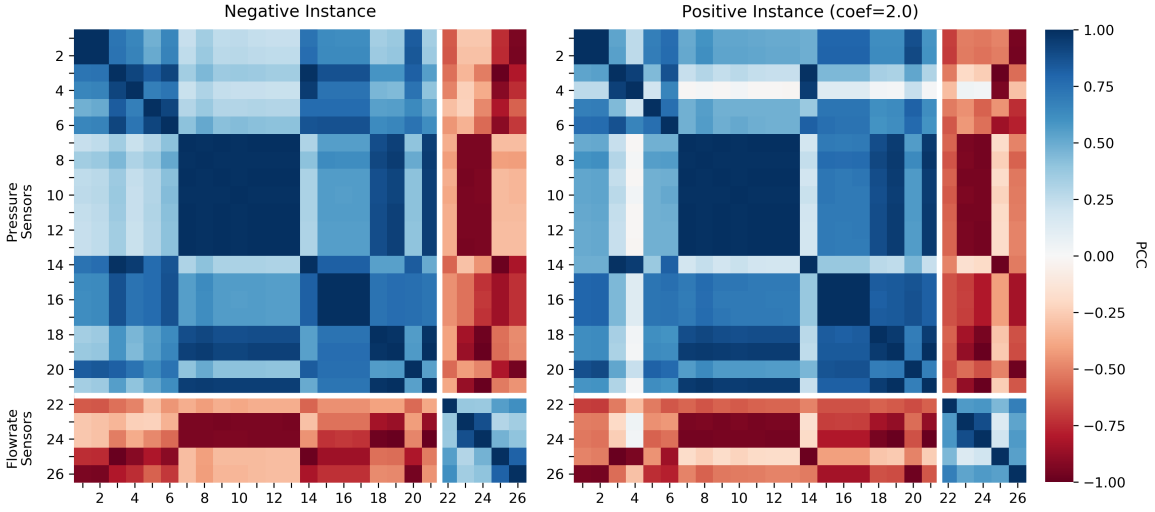

Figure 5.5: PCC heatmaps of two synthetic instances from chunk 702 using a time window of 40 time points.

Considering the 13 sensors in Infraquinta's WDN, our new real dataset has a total of ${}^{9}C_{2} = 36$ features. In Figure 5.7 we can see that the overall correlations are not as strong as in the synthetic dataset. We also noticed that DCCA seems to consider most sensors of the same type as directly correlated and sensors of different types as inversely correlated. However, when using PCC, these correlations are not as clear and seem to be the opposite. Regarding their patterns, although we can detect some similarities between DCCA and PCC heatmaps, it is not as apparent as in the synthetic heatmaps. Between the negative and the positive instance, while most correlations became weaker in PCC, some grew stronger in DCCA. Overall, DCCA's behavior in the real dataset is more similar to the synthetic one.

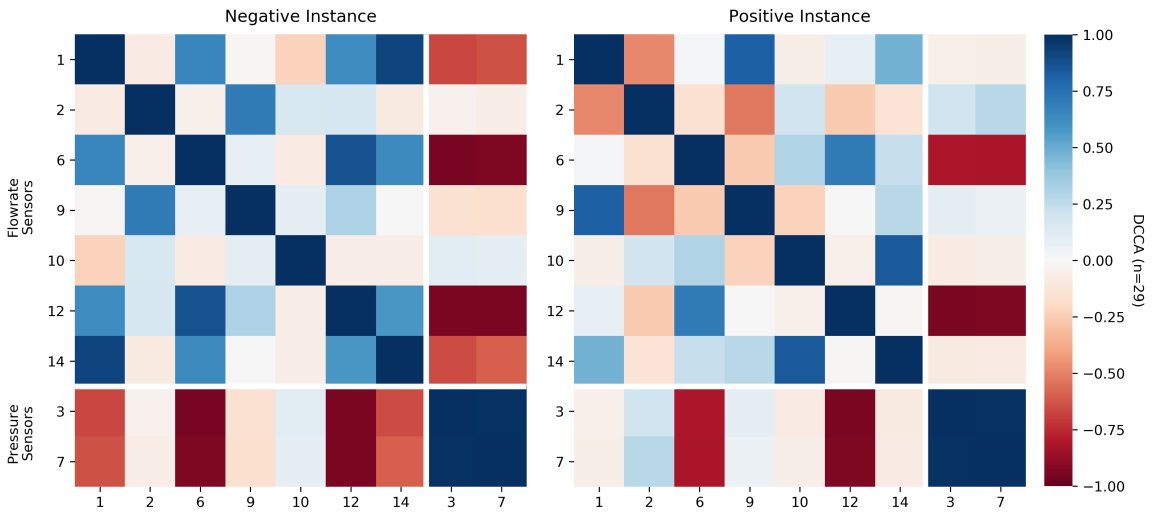

Figure 5.6: DCCA heatmaps of two real instances from February 7, 2017, using a time window of 120 minutes. The positive instance occurs between 10:00 and 12:00, while the negative one occurs between 14:45 and 16:45.

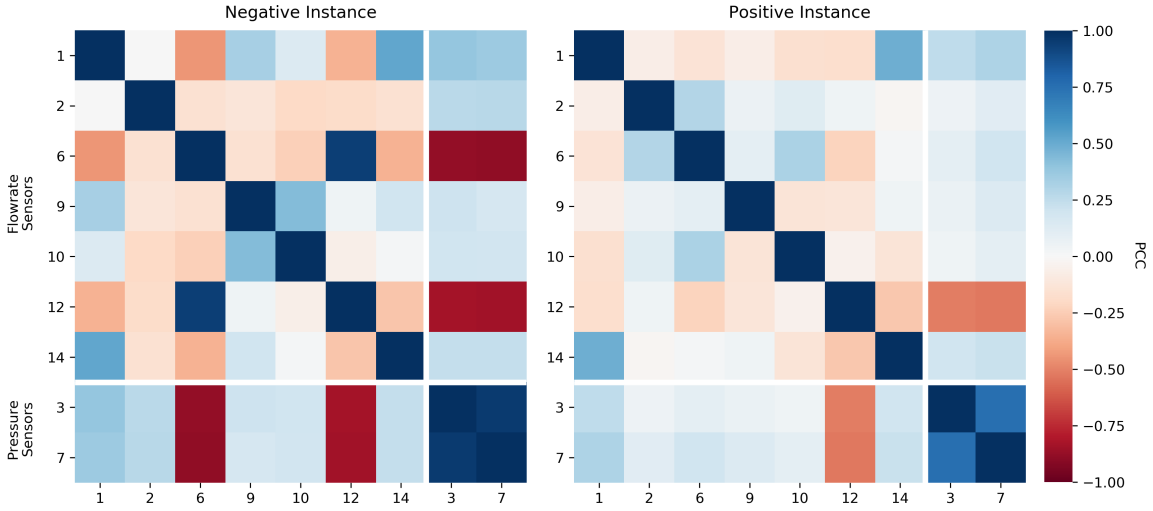

Figure 5.7: PCC heatmaps of two real instances from February 7, 2017, using a time window of 120 minutes. The positive instance occurs between 10:00 and 12:00, while the negative one occurs between 14:45 and 16:45.

5.3 Correlation Over Time

Another critical element to assess is how correlation evolves, especially before and during the leakage. To understand how the correlation changes over time in our synthetic setting, we created a sliding window that moves over the time series in intervals of one time point (10 minutes). Figure 5.8 shows the DCCA and PCC over time in three selected pairs from the same chunk. Before the leakage, DCCA and PCC remained between 0.5 and 1, negative or positive, depending on whether the pair is positively or negatively correlated. Moments after the leakage occurred, the correlations got weaker and closer to zero until they started getting stronger again. We can infer that this point corresponds to when the time window includes more leakage points than non-leakage points. Figure 5.8 also shows that DCCA oscillates a lot more than PCC, causing the impact of the leakage to be more apparent in DCCA.

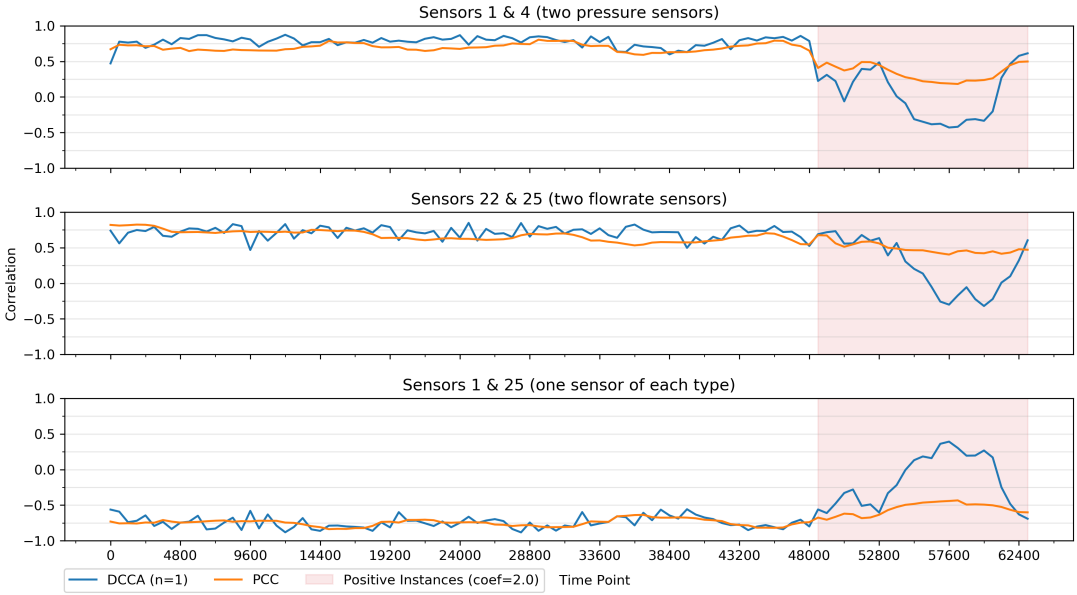

Figure 5.8: DCCA and PCC over time in three selected pairs from chunk 702 using a time window of 40 time points.

Regarding the real setting, as Figure 5.9 shows, the behavior of DCCA and PCC is very similar. Both methods are very unstable before the leakage, which is probably an indication that these methods can not accurately quantify the correlation value between time series due to the large volume of noise present. Although the differences between the negative and the positive instances are not as striking as in the

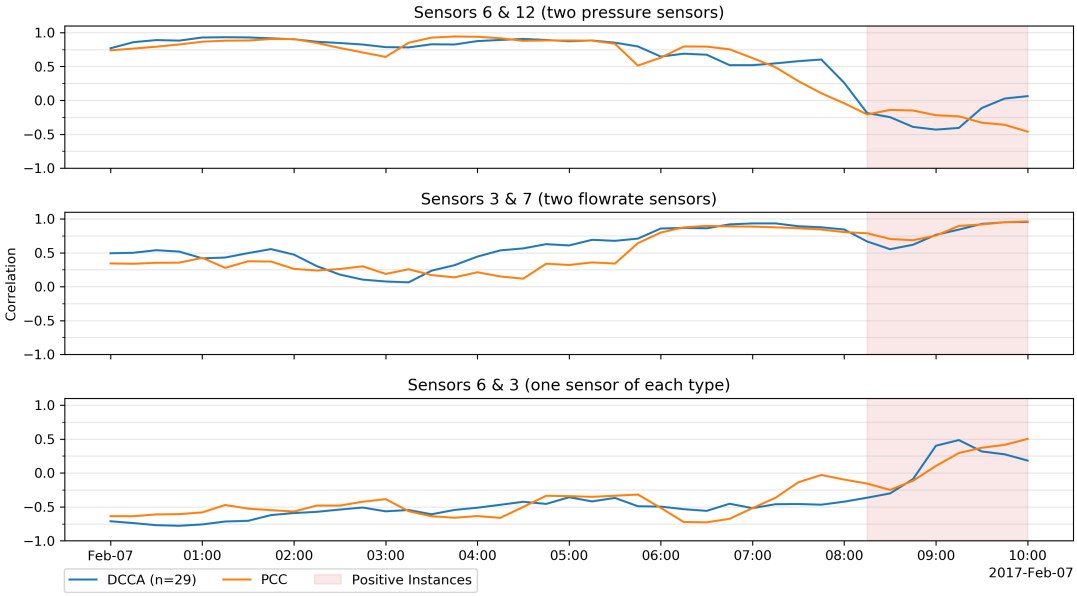

Figure 5.9: DCCA and PCC over time in three selected pairs from February 7, 2017, using a time window of 120 minutes.

synthetic dataset, we can still notice a subtle change after the leakage. Since PCC has not outperformed DCCA so far, and to make the analysis less exhaustive, the next sections of this chapter will only include the results obtained through DCCA.

5.4 Correlation in Small Leakages

For the experiments in the synthetic setting, we have been using a leakage coefficient of 2.0, which makes it a considerable leakage in size. However, since smaller leakages can also happen in our WDN, it is vital to understand how DCCA responds to it. Therefore, we plotted the DCCA values for six different leakage coefficients, as presented in Figure 5.10. As we can see, the DCCA value for coefficients below 0.5 is very

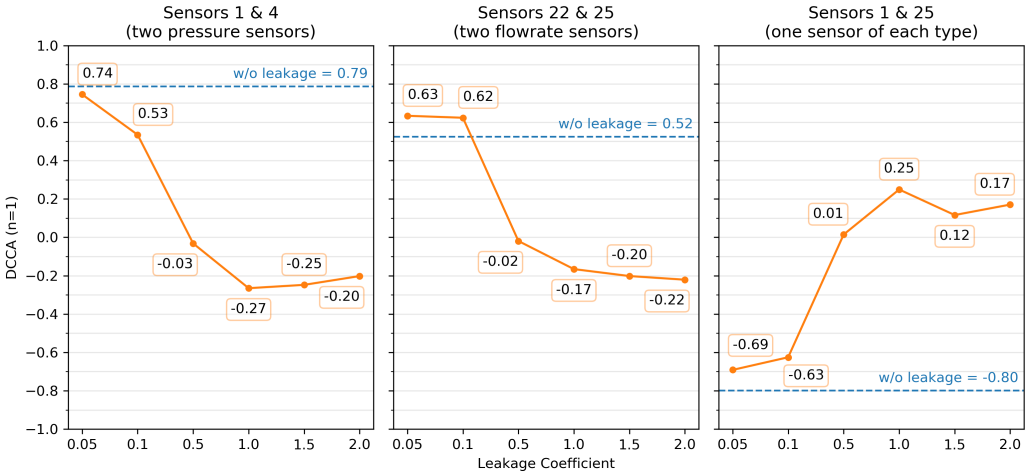

Figure 5.10: DCCA variation with the leakage coefficient in three selected pairs from chunk 702 using a time window of 40 time points.

close to the value obtained without a leakage. However, the correlation weakens considerably for leakages above 0.5. As expected, larger leakages seem to cause a higher disruption in DCCA than the smaller ones. Consequently, smaller leakages may be more difficult to detect than larger ones. Although we can not perform this analysis for the real setting because we do not have information about the leakage sizes, it is expected for smaller leakages ($coef \leq 0.1$) to go undetected.

5.5 Time Window Size

Until now, we have been using a time window of 40 time points for the synthetic setting, which is approximately 6h40. Figure 5.11 helps us understand how DCCA is affected by different time windows between 16 and 40 time points. Through the analysis of the positive and the negative instances, we can see that the difference between them seems considerably random until 32 time points. Around that period, DCCA values start diverging, peaking at 40 time points. One possible explanation is that DCCA could not accurately identify the degree of correlation with less than 32 points. Lastly, since the 40 time point window showed the largest correlation difference between instances, it may contribute to better results in the classification step than the others.

Regarding the real setting, we have been using a time window of 120 minutes. Figure 5.12 shows how DCCA is affected by four different time windows between 60 and 240 minutes. Although the 60 minute time window fluctuates a lot more than the others, all of them behave similarly over time. We can also see that the smaller its size, the greater its fluctuation. Focusing on the time window of 60 minutes, we

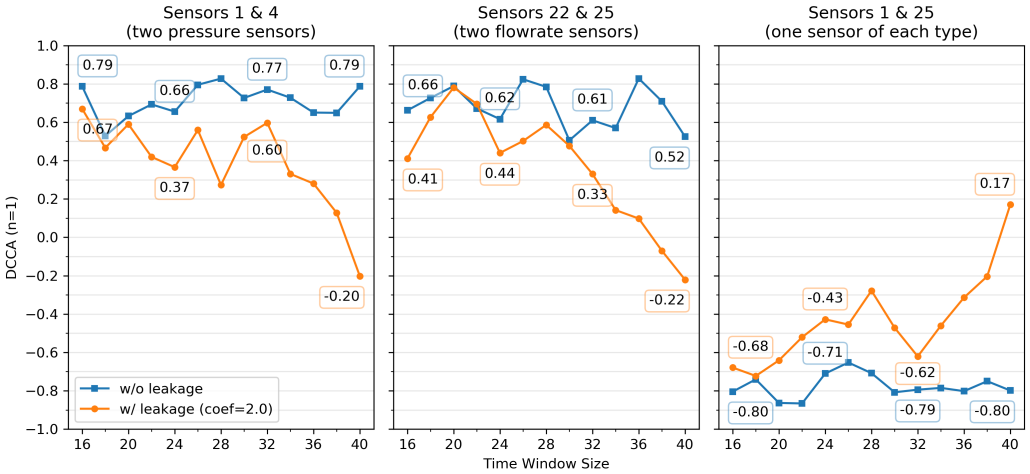

Figure 5.11: Impact of time window size on DCCA in three selected pairs from chunk 702.

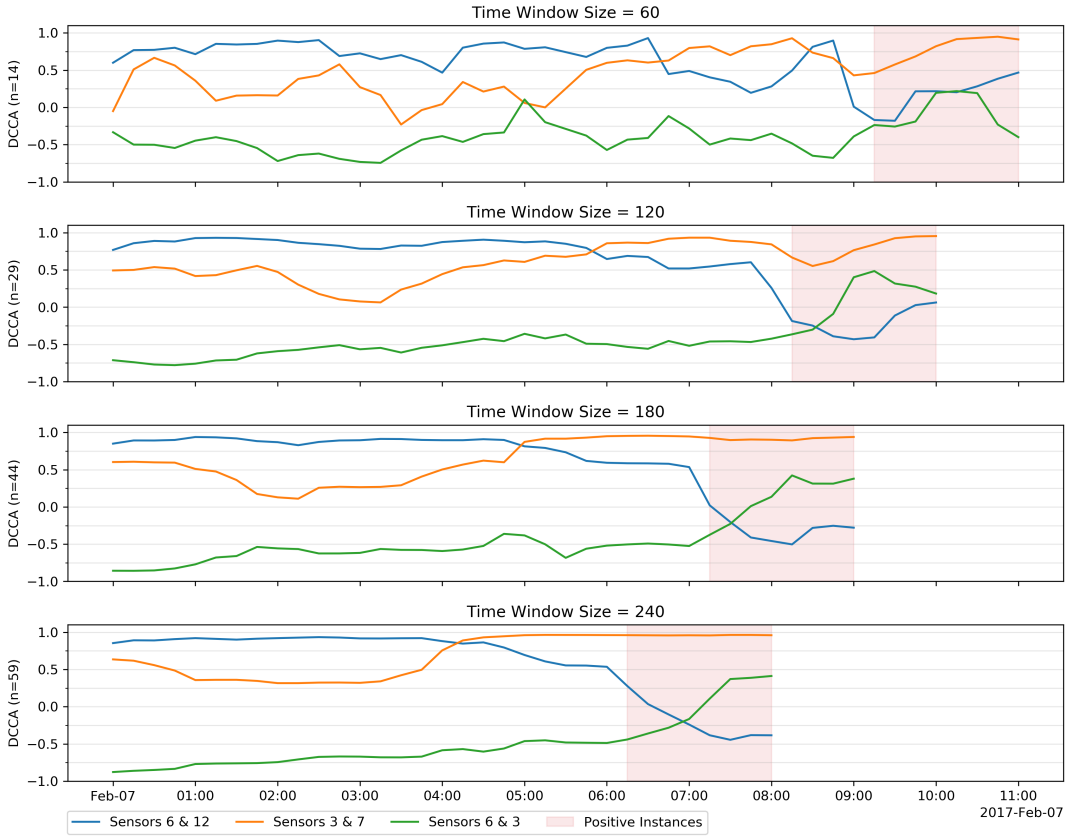

Figure 5.12: Impact of time window size on DCCA in three selected pairs from from February 7, 2017.

can see that too much variation disguises the leakage. Contrarily, 240 minutes causes so much stability that the correlation between sensors 3 and 7 does not change during the leakage. Lastly, the time window of 120 minutes seems to be very balanced, i.e., it does not fluctuate as much as one of 60 minutes but varies enough to let us still notice the difference between negative and positive instances.

5.6 DCCA Parameterization

As detailed in section 3.2.2, DCCA has a parameter n that affects the size of the overlapping boxes and directly affects its results. Therefore, it is important to assess its impact on both settings. Considering that $1 \leq n < T$, where T is the size of the time window, we plotted the DCCA for n values between 1 and 39 points, as Figure 5.13 shows. We can see that when $n = 1$, the difference between the DCCA values

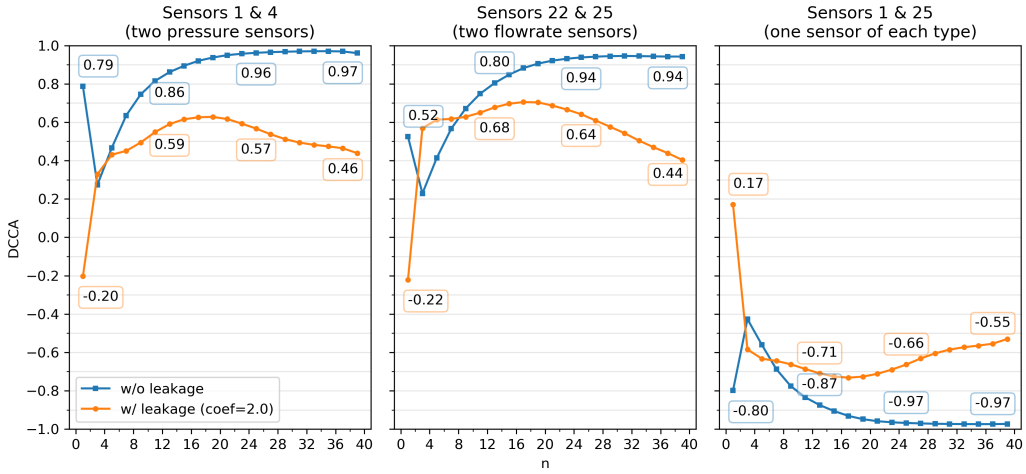

Figure 5.13: Impact of parameter n on DCCA in three selected pairs from chunk 702 using a time window of 40 time points.

for the negative and the positive instances reaches its peak. For $n > 1$, the difference between instances slowly grows until $n = 39$, its best value after $n = 1$. Therefore, $n = 1$ may contribute to better results in the classification step than any other value of n.

To understand how n affects DCCA in the real setting, we chose four different values so that the boxes' sizes would be $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$, and $\frac{1}{5}$ of our time window. Figure 5.14 shows how the correlation of each

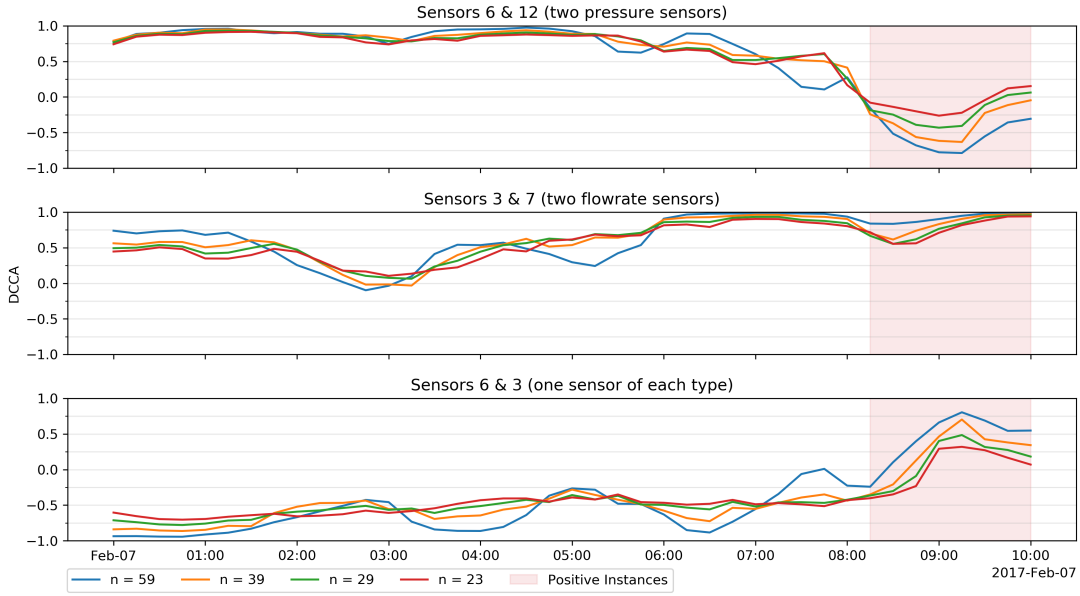

Figure 5.14: Impact of parameter n on DCCA in three selected pairs from February 7, 2017, using a time window of 120 minutes.

pair is affected by each n value. Although all four values of n seem to follow the same pattern, $n = 59$ stands out from the others since it fluctuates a lot more. The other three do not fluctuate as much as $n = 59$ but vary enough to let us notice the difference between negative and positive instances.

Chapter 6

Results: Predictive Setting

This chapter assesses the predictive ability of detecting leakages in controlled and real settings, presenting a comprehensive analysis of the hyperparameterization steps performed to identify the most suitable leakage dynamic predictors for Infraquinta's WDN. In section 6.1, we learn how a classifier performs on a WDN under ideal conditions. Then, in section 6.2, we use that knowledge to (1) overcome the classification problems caused by real conditions, and (2) find the best hyperparameters for our final classifiers.

6.1 Artificial Water Distribution Network

This section focuses on understanding how a classifier performs on a WDN under ideal conditions. Therefore, and as explained in section 4.2.1, we used a dataset constructed with data generated by an artificial EPANET model of Infraquinta's network. This synthetic dataset, as Table 6.1 shows, has a total of 37392 instances, where half is negative and the others are positive. The positive instances are equally

	Negative Instances	Positive Instances	Total
Training Set	14956	14957	29913
Test Set	3740	3739	7479
Total	18696	18696	37392

Table 6.1: Summary of the synthetic dataset.

distributed between 6 different leakage sizes, i.e., 3116 instances per leakage coefficient. Additionally, we used 80% of the dataset for training and the remaining 20% to perform the final evaluation.

Regarding the next steps, section 6.1.1 presents the initial results of three classifiers in the synthetic setting using all features. Then, in section 6.1.2, we assess how decreasing the number of features affects their performance. Section 6.1.3 focuses on choosing the ideal time window size. Next, section 6.1.4 centers around finding an appropriate threshold and studying its impact on small leakages. Lastly, section 6.1.5 reveals the results obtained with the test set.

6.1.1 Initial Results

To get an overall idea of the classifiers' performance in our synthetic setting, we started by gathering the results obtained using DCCA and PCC as the correlation methods. Additionally, to understand how the sensor type affects the performance, we also included the results for when the dataset features include either pressure or volumetric flowrate sensors. Accordingly, Table 6.2 presents the mean results for a 5-fold cross-validation on the training set.

Classifier	Correlation Method	TWS	Sensors	#F	$Prec_P$	$Recall_P$	F_P
Naive Bayes	DCCA ($n = 1$)	40	Both	325	0.88	0.71	0.78
			Pressure	210	0.88	0.69	0.77
			Flow	10	0.83	0.59	0.69
	PCC	40	Both	325	0.93	0.64	0.76
			Pressure	210	1.00	0.61	0.76
			Flow	10	0.84	0.49	0.62
SVM (Linear Kernel)	DCCA ($n = 1$)	40	Both	325	1.00	1.00	1.00
			Pressure	210	1.00	0.99	1.00
			Flow	10	0.82	0.64	0.72
	PCC	40	Both	325	1.00	0.97	0.99
			Pressure	210	1.00	0.93	0.97
			Flow	10	0.79	0.56	0.66
SVM (RBF Kernel)	DCCA ($n = 1$)	40	Both	325	1.00	0.75	0.86
			Pressure	210	1.00	0.75	0.86
			Flow	10	1.00	0.71	0.83
	PCC	40	Both	325	0.94	0.70	0.80
			Pressure	210	0.99	0.71	0.83
			Flow	10	1.00	0.71	0.83

Table 6.2: Initial results of three classifiers in the synthetic setting using a 5-fold cross-validation on the training set. Note that TWS and #F refer to the time window size and the number of features used, respectively.

Regarding NB, although PCC originates a higher $precision_P$, its $recall_P$ is much lower than DCCA's. Consequently, DCCA's F-measure is higher than PCC's, indicating that the overall combination of the $recall_P$ and the $precision_P$ is better. We can also see that the inclusion of the flowrate sensors lowered the number of false positives when using DCCA, improving its $precision_P$. In PCC's case, the $precision_P$ decreased, but since the $recall_P$ also increased, the overall performance did not change.

Regarding SVMs, we can see that DCCA provided better results than PCC. With a linear kernel, the inclusion of the flowrate sensors lowered the number of false negatives, improving the $recall_P$. However, with an RBF kernel, the inclusion of both sensor types negatively affected the $precision_P$ when using PCC. In general, the linear kernel shows better results than the RBF.

On the overall performance of the classifiers, NB performs worse than our two SVMs, with the linear kernel showing the best results. As expected and per our results in sections 5.2 and 5.3, we can infer that DCCA provides better results than PCC. Hence, the next sections will only include the results obtained through DCCA. Finally, volumetric flowrate sensors got worse results than the pressure ones,

but we believe the cause to be their low number and poor distribution. Nevertheless, using both types of sensors is usually better than using only one. It is also important to note that all iterations of the 5-fold cross-validation obtained similar results, meaning that the classifiers do not seem to be overfitting the data. Additionally, unless otherwise stated, the reader can always assume that for this setting.

6.1.2 Feature Selection

The goal of this section is to assess if we can reduce the number of features (pairs of sensors) without negatively affecting the classifiers' performance on our synthetic setting. Hence, for each classifier, we plotted three performance measures along with the top k features, starting from the top 15 to the top 100. The top k features were selected by our feature ranking method and Kruskal–Wallis H test.

Regarding NB, Figure 6.1 shows that when using our feature ranking method, the performance improves considerably around top 30, peaking at 43. Note that NB's performance using these 43 features is

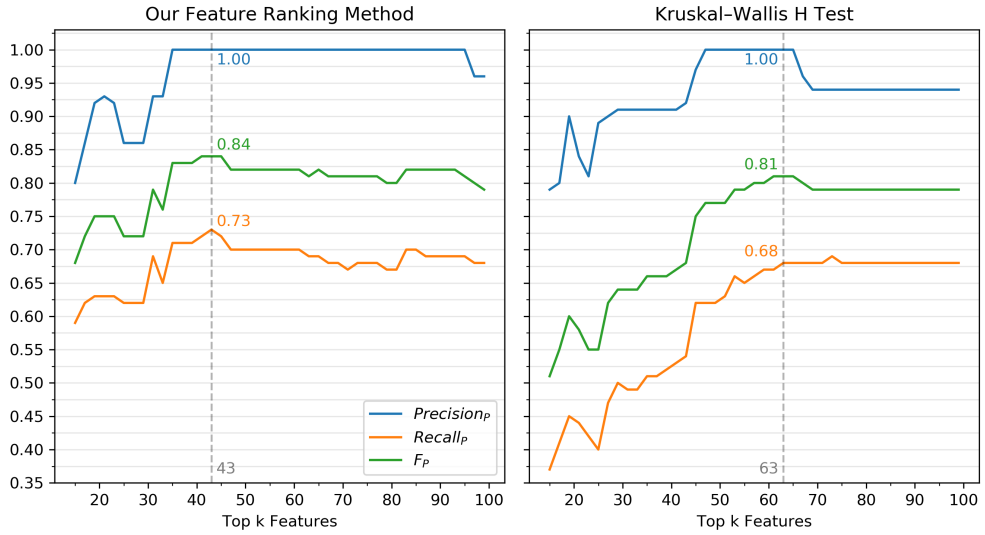

Figure 6.1: NB's performance in the synthetic setting along with the top k features selected by two feature ranking strategies. Note that we used a 5-fold cross-validation on the training set, DCCA ($n = 1$) as the correlation method, and a time window of 40 time points.

better than with 325, meaning that some features could be misleading and uninformative. As we can see, its performance drops soon after that. With the Kruskal–Wallis H test, we also detected an improvement around the top 30, peaking at 63. Contrarily to NB, this method did not obtain better results than if we used all 325 features. Moreover, our method immediately identified an initial set of features that provided reasonable results. Overall, our feature ranking method helped NB converge much faster to better results than the Kruskal–Wallis H test.

Focusing on Figure 6.2, SVM's performance with a linear kernel also improves suddenly around top 30. Although both ranking methods continue to converge to one, they never reach it using the top 100 alone. Nevertheless, the Kruskal–Wallis H test still manages to deliver competitive results using 83 features. Our ranking method also managed to obtain good results, but Kruskal–Wallis H test converged faster. We can conclude that although no ranking method reached the results obtained using 325 features, Kruskal–Wallis H test performed better than our feature ranking method.

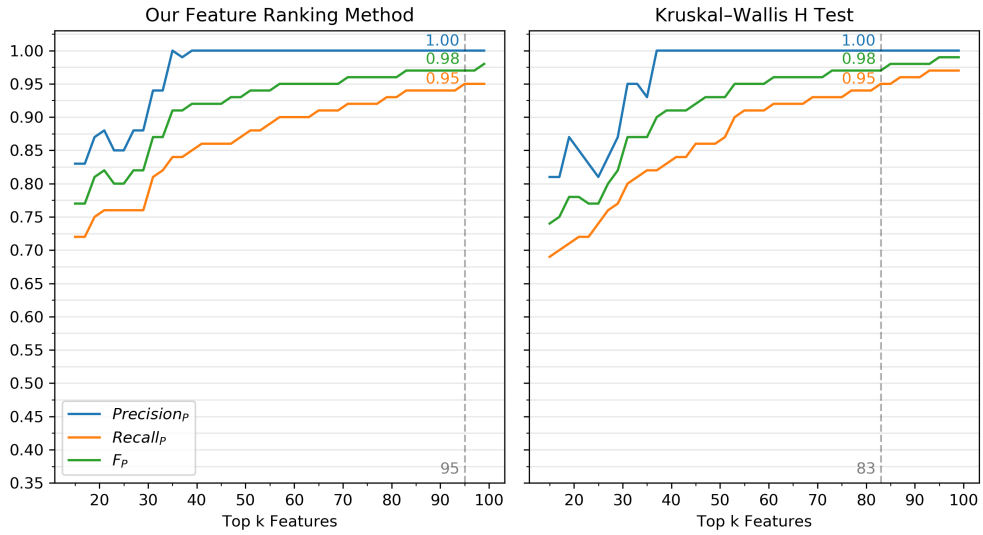

Figure 6.2: SVM's performance with a linear kernel in the synthetic setting along with the top k features selected by two feature ranking strategies. Note that we used a 5-fold cross-validation on the training set, DCCA ($n = 1$) as the correlation method, and a time window of 40 time points.

Concerning the SVM with the RBF kernel, Figure 6.3 shows that the results obtained with these two ranking methods are very distinct. Using our feature ranking method, the performance of the

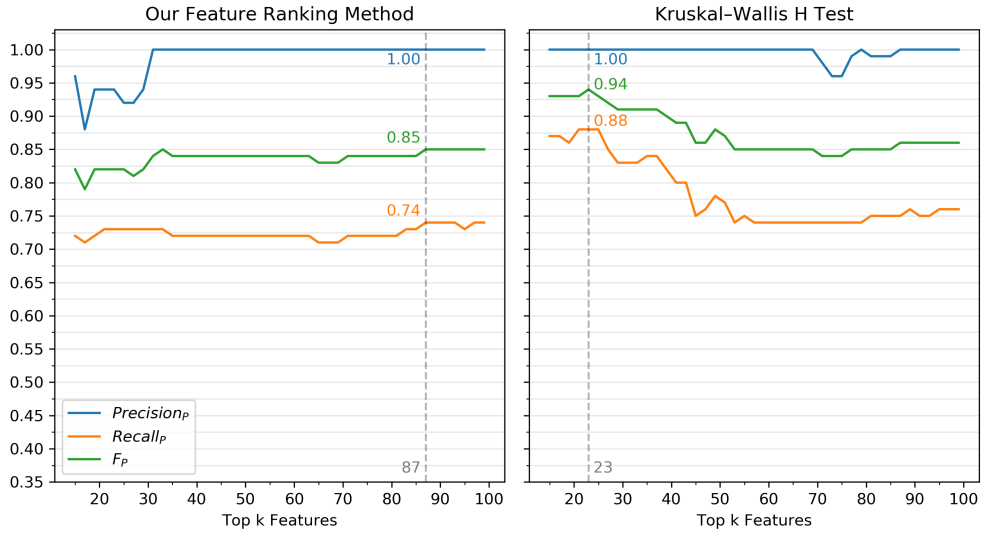

Figure 6.3: SVM's performance with a RBF kernel in the synthetic setting along with the top k features selected by two feature ranking strategies. Note that we used a 5-fold cross-validation on the training set, DCCA ($n = 1$) as the correlation method, and a time window of 40 time points.

classifier improves around the top 30. However, it remains almost stable until it peaks at 87. Although its performance is not as good as with a linear kernel, it still manages to get very close to the results obtained using 325 features. Contrarily, the Kruskal–Wallis H test identified a set of only 23 features that provided better results than using all of them. However, as in NB, using more features immediately caused the $recall_P$ to drop.

About the overall results using these two feature ranking methods, the classifiers' performance seems to improve around the top 30. Although our feature ranking method only worked better than the

Kruskal–Wallis H test on NB, it still managed to improve its performance. Additionally, Kruskal-Wallis considerably improved the SVM's performance when using an RBF kernel, but it could not match the results obtained with a linear kernel. Finally, we conclude that (1) both methods work depending on the classifier and (2) more features do not always result in the best performance. To continue this analysis, we decided to choose only one SVM kernel to compare to NB. Since the RBF kernel did not outperform the linear one, the next sections will not include an SVM with this kernel.

6.1.3 Time Window Size

This section focuses on finding the ideal size of the time window and how it affects the performance of our classifiers. Until now, we have been using a time window of 40 time points for the synthetic setting, which is approximately 6h40. Note that for NB, we used the top 43 features selected by our feature ranking method, and for the SVM with the linear kernel, we used the top 83 features selected by Kruskal–Wallis H test. As in section 5.5, we used time windows between 16 and 40 time points, as Figure 6.4 shows.

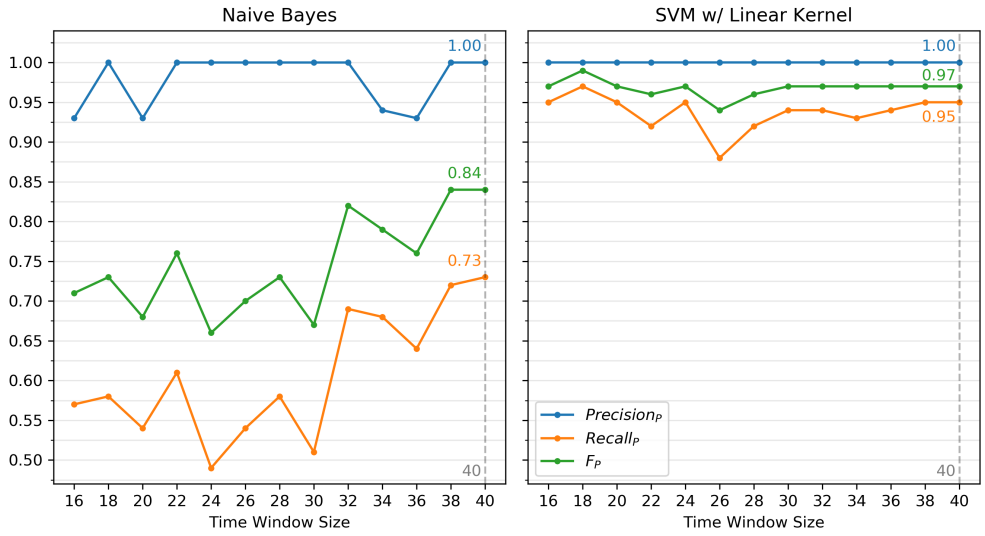

Figure 6.4: Classifiers' performance in the synthetic setting along with different time window sizes. NB uses the top 43 features selected by our feature ranking method, while the SVM with the linear kernel uses the top 83 features selected by Kruskal–Wallis H test. Note that we used a 5-fold cross-validation on the training set and DCCA ($n = 1$) as the correlation method.

Regarding NB, although we can see that the $precision_P$ oscillates when using some window sizes, it is much more stable than the $recall_P$. The $recall_P$ fluctuates substantially, but we can still see an increase around a 30 time point window. Suddenly, both measures drop soon after that, but not enough to prevent NB from peaking at 40.

SVM's $recall_P$ is also more unstable than $precision_P$. Nevertheless, it starts converging after a 26 time point window, peaking at 40. Unexpectedly, the results obtained using a time window of only 18 points are as good as using one with 40, which goes against the analysis performed in section 5.5. Since the $recall_P$ drops soon after that, it may be an indication that it is just a coincidence.

We can conclude that increasing the size of the time window does not always enhance the classifiers' performance. As expected and per the results obtained in section 5.5, we see an improvement in both

classifiers around 30 time points. Unlike NB, SVM seems to be able to adapt well to different time windows. Overall, using a time window of 40 time points obtained the best results for both classifiers.

6.1.4 Threshold Adjustment

As mentioned in section 4.3.2, binary classifiers usually use a threshold to determine if the instances are positive or negative. In both NB's and SVM's case, their default threshold is zero. Since that threshold might be preventing them from achieving optimal results, this section focuses on finding an appropriate threshold and studying its impact on small leakages.

Receiver Operating Characteristics Analysis

To find the optimal threshold ρ for each classifier, we plotted the TPR and the FPR over all possible values of the threshold, resulting in the Receiver Operating Characteristics (ROC) plots shown in Figure 6.5.

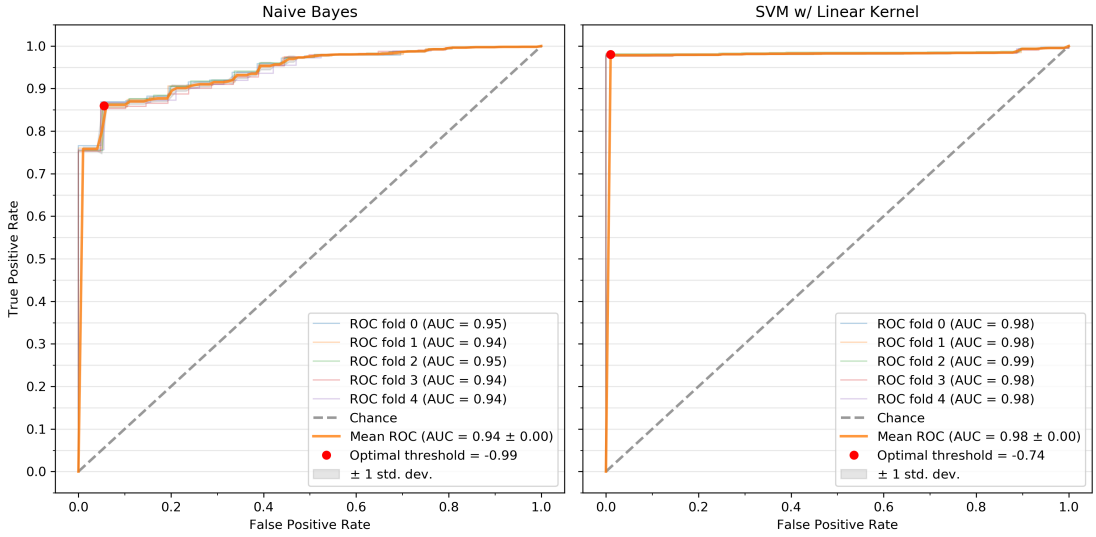

Figure 6.5: Classifiers' ROC plot in the synthetic setting. NB uses the top 43 features selected by our feature ranking method, while the SVM with the linear kernel uses the top 83 features selected by Kruskal–Wallis H test. Note that we used DCCA (n=1) as the correlation method and a time window of 40 time points.

Regarding NB's ROC plot, the ROC curves of the five folds result in an optimal threshold of -0.99. As we can see, the optimal threshold corresponds to the point closer to the upper left corner, thus maximizing TP and minimizing FP. This threshold in particular indicates that NB will only classify an instance as negative if the difference between the probability of being negative and the probability of being positive is larger than -0.99. Note that this only happens when the probability of being negative is 1 and the probability of being positive is 0, which inevitably increases the number of FP.

Concerning the SVM, its optimal threshold corresponds to -0.74. Unlike NB, SVM's threshold does not have a value limit. Note that SVM's curve is closer to the upper left corner than NB's, resulting in a larger AUC. Therefore, after finding the optimal thresholds, SVM's performance remains better than NB's without the need of misclassifying negative instances.

Threshold Values

Although the ROC plot allows us to quickly understand what the best performance a classifier can achieve is, it hides how its performance varies along with different threshold values. Hence, Figure 6.6 shows the impact of threshold variation on our classifiers. Since SVM's threshold values are as large as the

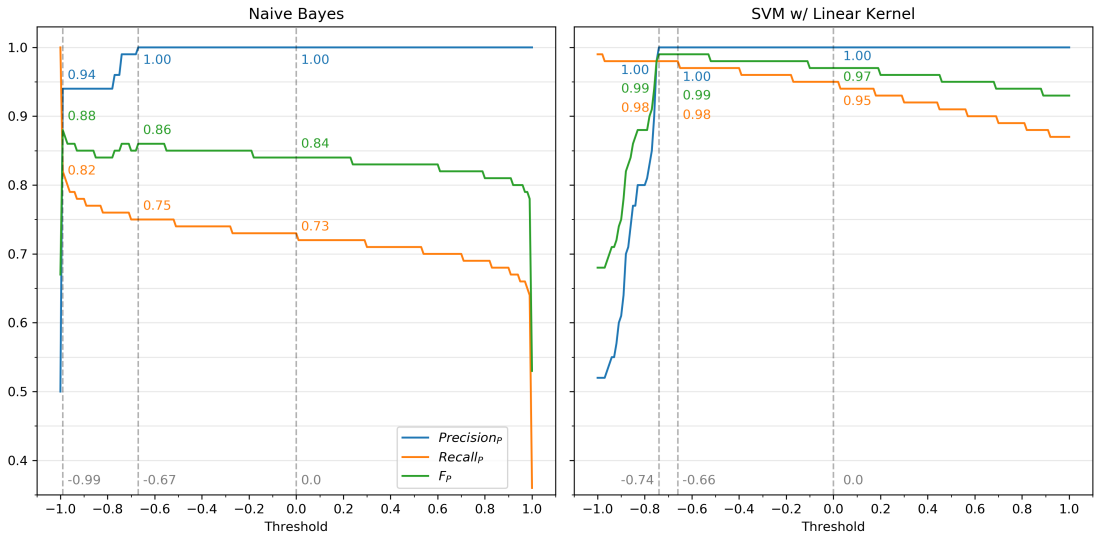

Figure 6.6: Classifiers' performance in the synthetic setting along with different thresholds. NB uses the top 43 features selected by our feature ranking method, while the SVM with the linear kernel uses the top 83 features selected by Kruskal–Wallis H test. Note that we used a 5-fold cross-validation on the training set, DCCA (n=1) as the correlation method, and a time window of 40 time points.

instances' distance to the separating hyperplane, we only plotted the performance measures along with threshold values between -1 and 1.

As expected, when we lower the threshold, more instances are classified as positive and the $recall_P$ increases. In particular, we can see that NB performance slowly improves until $\rho = -0.67$. After that, the $recall_P$ rapidly increases and the $precision_P$ drops, meaning that the classifier is starting to misclassify negative instances in order to correctly classify the positive ones. Then, when $\rho = -1$, NB classifies all instances as positive, causing the $recall_P$ to increase to 1 and the $precision_P$ to drop to 0.5. Since $\rho = -0.99$ increases the number of FP, another option would be to use a threshold of -0.67. Although the results are not as good as with the optimal threshold, $\rho = -0.67$ still increases the $recall_P$ and prevents instances from being classified as negative if their negative class score is low.

Like NB, SVM's $recall_P$ also increases when we lower the threshold. Then, after peaking at the optimal threshold $\rho = -0.74$, the $precision_P$ rapidly drops, indicating that some positive instances' scores are close to the majority of the negative ones. Consequently, to correctly classify those positive instances, SVM also has to classify the negative ones as positive, generating a large number of FP. Lastly, note that $\rho = -0.66$ achieves the same results as the optimal threshold $\rho = -0.70$. So, we can obtain the same performance without lowering the threshold as much.

Identification of Small Leakages

Until now, we focused on the overall performance of the classifiers without assessing its impact on small leakages. Therefore, we decided to use boxplots to understand the to understand how our classifiers respond to different leakage sizes. Boxplots show an aggregate statistical summary of their values through their quartiles [86]. Hence, they use the median (50% point), the lower and upper quartiles (25% and 75% points), and the upper and lower whiskers. The longer the spaces between the different box parts, the greater its degree of dispersion. Box plots also allow us to understand the degree of skewness, i.e., how symmetrical the distribution is. Additionally, all points outside the whiskers' boundaries are considered outliers. For readability purposes, we did not visually include them in Figure 6.7. Note that the box plots with a leakage coefficient of zero correspond to the box plots of the negative instances.

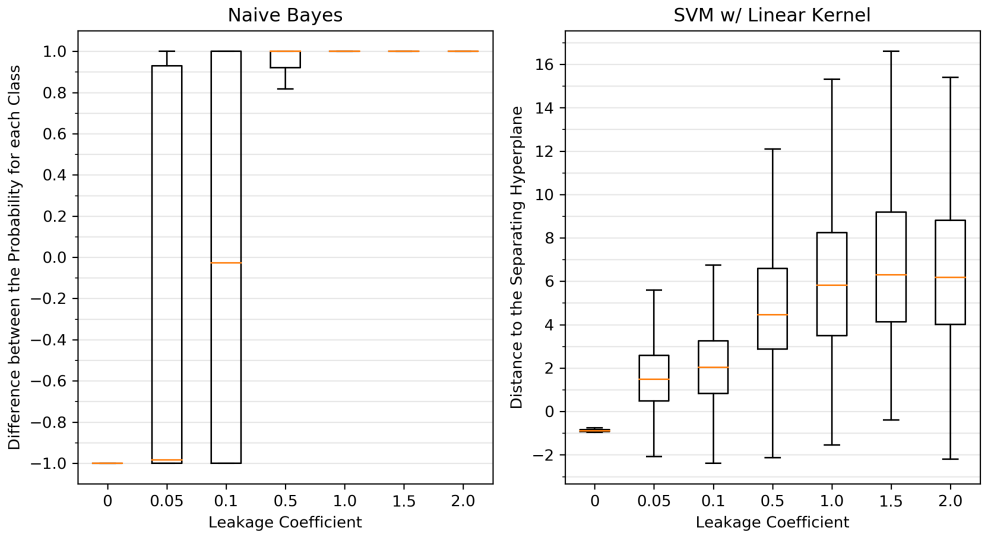

Figure 6.7: Classifiers' box plots of the instances' scores for the synthetic setting. Note that NB uses the top 43 features selected by our feature ranking method, while the SVM with the linear kernel uses the top 83 features selected by Kruskal–Wallis H test. Moreover, we used a 5-fold cross-validation on the training set, DCCA (n=1) as the correlation method, and a time window of 40 time points.

Regarding NB, the majority of the positive instances' scores are around 1, i.e., the highest score possible. The only exception is when the leakage coefficients are 0.05 and 0.1, meaning that NB is not as confident about their true class as for larger coefficients. Looking at the instances with these two coefficients, we can see that their median scores are not as close to 1 as the others. Furthermore, the median score for instances with a coefficient of 0.005 is almost -1, proving that these instances are very similar to the negative ones and harder to classify. Lastly, the score of most negative instances is around -1, revealing NB's certainty about their predicted class.

In SVM's case, Figure 6.7 clearly shows that the lower the coefficient of a positive instance, the closer its score is to zero. Like in NB, it proves that SVM is not as confident about the true class of these instances as it is for the others. Once again, the majority of negative instances have similar scores. However, some positive instances were able to get even lower scores, proving that these are hard to differentiate from the negative ones.

Overall, both classifiers exhibit confidence in the classification of negative instances and positive

instances with coefficients higher than 0.1. When dealing with smaller leakages, both classifiers have difficulty differentiating them from those without it. It is also worth noting that the problem with the identification of some leakages might not be exclusively due to their size but also because of their location.

To conclude the analysis of the threshold's impact on small leakages, we plotted the percentage of misclassified instances by their leakage coefficient, as figure 6.8 shows. As expected and per our results

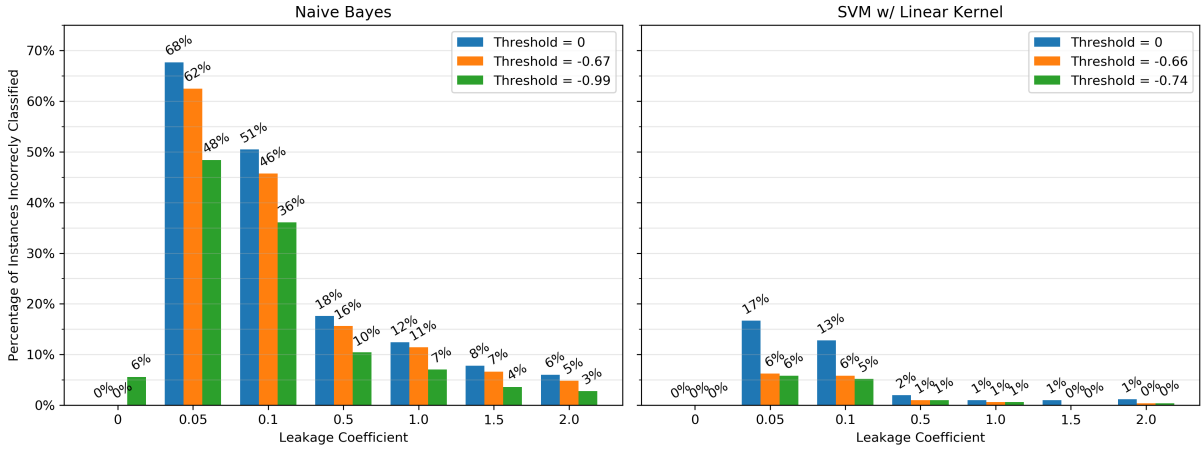

Figure 6.8: Percentage of instances incorrectly classified using three different thresholds. NB uses the top 43 features selected by our feature ranking method, while the SVM with the linear kernel uses the top 83 features selected by SelectKBest with the Kruskal–Wallis H test. Note that we used a 5-fold cross-validation on the training set, DCCA ($n = 1$) as the correlation method, and a time window of 40 time points.

in section 5.4, the percentage of FN is much larger for instances with coefficients of 0.05 and 0.1 than for others. As we have seen in Figure 6.7, NB has difficulty in differentiating these instances from the negative ones. Regarding the thresholds, one of -0.99 obtains the best results and reduces the number of false positives by around 20% in these two types of leakages. However, as we have also seen in Figure 6.6, it increases the number of FP. A more modest alternative would be to choose a threshold of -0.67. Although its impact in the reduction of FN is not as considerable, it still helps decreasing the total of FN and and prevents instances from being classified as negative if their negative class score is low. Concerning the SVM, it also has a slight difficulty in identifying leakages with coefficients of 0.05 and 0.1. Additionally, we can see that there is not a significant difference between a threshold of -0.74 and -0.66. Finally, considering these results, we decided to move to the next section using a threshold of -0.67 for NB and one of -0.66 for the SVM.

6.1.5 Test Set Results

In the previous sections, we used a 5-fold cross-validation on the training set to assess the performance of our classifiers and tune its hyperparameters. Since all iterations and experiments were successful, this section focuses on presenting the final evaluation, i.e., the results obtained with the test set.

Table 6.3 shows the overall results of NB and SVM with a linear kernel using the hyperparameters defined earlier. Note that since the results obtained using the test set are equal to the mean results of the 5-fold cross-validation on the training set, we decided to show only one table. Accordingly, it proves

Classifier	Correlation Method	TWS	Feature Ranking Method	#F	Threshold	$Prec_P$	$Recall_P$	F_P
Naive Bayes	DCCA ($n = 1$)	40	Our Feature Ranking Method	43	-0.67	1.00	0.75	0.86
SVM (Linear Kernel)	DCCA ($n = 1$)	40	Kruskal–Wallis H test	83	-0.53	1.00	0.98	0.99

Table 6.3: Results obtained by NB and SVM with a linear kernel on the synthetic test set.

that our classifiers are not overfitting and are generic enough to accommodate small changes in the data. Furthermore, both classifiers reached maximum $precision_P$, meaning that all instances classified as positive are indeed positive. However, as our previous experiments showed, it is more difficult for NB to identify leakages than SVM, failing in recognizing 25% of them.

To get a detailed view of the results, we plotted the percentage of misclassified instances by their leakage coefficient, as depicted in Figure 6.9. As expected and per the results obtained in the previous

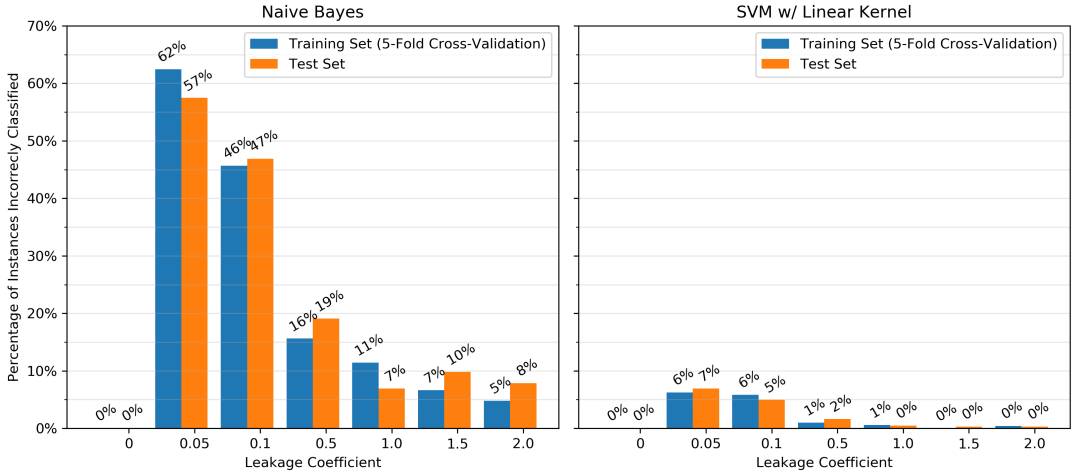

Figure 6.9: Percentage of instances incorrectly classified on the training and test set. NB uses the top 43 features selected by our feature ranking method, while the SVM with the linear kernel uses the top 75 features selected by Kruskal–Wallis H test. Note that we used a 5-fold cross-validation on the training set, DCCA (n=1) as the correlation method, and a time window of 40 time points.

experiments, NB is not very good at identifying small leakages, i.e., positive instances with coefficients of 0.05 and 0.1. Furthermore, the differences between the results obtaining using the training set and the test set for each coefficient are not significant. Regarding the SVM, the test set confirmed that the classifier has a slight difficulty in differentiating smaller leakages from negative instances. Moreover, as in NB, the results for each coefficient are very similar to the ones obtained with the training set. Overall, the test set confirms that (1) some positive instances are difficult to differentiate from negative ones, and (2) although SVM is much better at identifying leakages, NB still manages to identify more than 80% of the positive instances with coefficients larger than 0.1.

6.2 Real Water Distribution Network

Contrary to section 6.1, here we focus on understanding how a classifier performs under real conditions. Therefore, and as explained in section 4.2.1, we used a dataset constructed with real data from Infraquinta's network. This dataset, as Table 6.4 shows, has a total of 33529 instances, where 99.6% are negative and the remaining 0.4% are positive. Due to the severe class imbalance, the number of folds

	Negative Instances	Positive Instances	Total
Training Set	30617	119	30736
Test Set	2784	9	2793
Total	33401	128	33529

Table 6.4: Summary of the real dataset.

used is equal to the number of leakages, ensuring one leakage in the validation set, while the others are used for training. Please note that one leakage is equivalent to more than one positive instance due to the sliding time windows. We also put one of the folds aside to use in the final evaluation.

Regarding the next steps, section 6.2.1 presents the initial results of three classifiers in the real setting using all instances. Section 6.2.2 studies the class imbalance by understanding how lowering the number of negative instances affects the classifiers. Then, in section 6.2.3, we assess how decreasing the number of features affects their performance. Section 6.2.4 focuses on choosing the ideal time window size. Next, section 6.2.5 centers around finding what the best performance our classifiers can achieve is and choosing an appropriate threshold. Lastly, section 6.2.6 reveals the results obtained with the test set.

6.2.1 Initial Results

As in section 6.1, we started by gathering the results obtained with three different classifiers using DCCA and PCC as the correlation methods. Accordingly, Table 6.5 presents the mean results for an 11-fold cross-validation on the training set. It is also important to note that we included two additional

Classifier	Correlation Method	TWS	#F	$Recall_N$	$Prec_P$	$Recall_P$	F_P	L
Naive Bayes	DCCA ($n = 29$)	120	36	0.99	0.00	0.00	0.00	0.00
	PCC	120	36	0.98	0.01	0.04	0.01	0.27
SVM (Linear Kernel)	DCCA ($n = 29$)	120	36	1.00	0.00	0.00	0.00	0.00
	PCC	120	36	1.00	0.00	0.00	0.00	0.00
SVM (RBF Kernel)	DCCA ($n = 29$)	120	36	0.80	0.00	0.06	0.00	0.36
	PCC	120	36	0.68	0.00	0.33	0.01	0.55

Table 6.5: Initial results of three classifiers in the real setting using an 11-fold cross-validation on the training set. Note that TWS and #F refer to the time window size and the number of features used, respectively. Lastly, L represents the portion of leakages detected.

performance measures, namely, the $recall_N$ and L. While the $recall_N$ allows us to understand the portion of the negative instances correctly classified, L represents the portion of leakages detected. For example, $L = 1$ indicates that all iterations of the 11-fold cross-validation correctly classified at least one positive

instance. Although the goal is to correctly classify all positive instances, we only need to identify one per iteration to detect a leakage.

Regarding NB, when using DCCA as the correlation method, it fails to identify all positive instances. When using PCC, although the results are not very different, it manages to detect three leakages. Note that the $recall_P$ is still very low, meaning that the model is not differentiating the positive instances from the negative ones.

In SVMs' case, when using a linear kernel, the classifier does not learn the difference between classes and misclassifies all positive instances. However, with an RBF kernel, the results look more promising. When using DCCA, the SVM detects four leakages but still fails to identify the majority of positive instances. With PCC, it detects five leakages and identifies 33% of positive leakages. However, the $recall_N$ dropped and is less consistent through all iterations.

Overall, it is difficult for all three classifiers to learn the difference between positive and negative instances. Consequentially, when the $recall_P$ increases, the $recall_N$ drops, meaning that the classifiers have to misclassify the negative instances to identify the positive ones. The SVM with an RBF kernel looks more promising, but its cross-validation results are not consistent through all iterations. Several problems may contribute to these results, including the severe class imbalance.

6.2.2 Reduction of Negative Instances

To understand how the class imbalance is affecting the classifiers, we plotted the previous performance measures along the reduction of negative instances. To achieve this, we started our experiments with a 10% random sample of negative instances from our training set (3062 instances), until we reached an almost balanced training set, i.e., a 0.5% random sample (153 instances). Note that the chosen samples may not be representative of the class, causing the loss of crucial information.

Regarding NB, Figure 6.10 shows that the classifier benefits from the reduction of the setting, especially when using PCC. Although the $recall_P$ never reaches the values obtained in the synthetic setting,

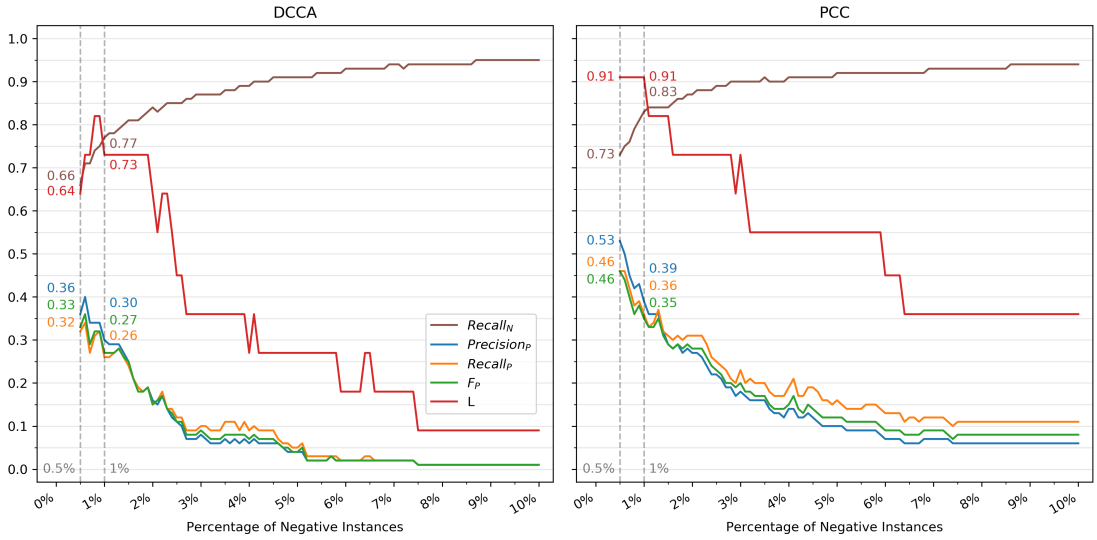

Figure 6.10: NB's performance along with the reduction of negative instances. Note that we used an 11-fold cross-validation on the training set and a time window of 120 minutes.

NB detected a leakage in 10 iterations out of 11. However, the number of TN decreases for the number of TP to increase, meaning that our classifier does not fully differentiate these instances. One additional note is that although NB technically performs better when the percentage of negative instances is 0.5%, the results in each iteration are less consistent than with a sample of 1% (306 instances).

As in NB, Figure 6.11 shows that SVM's performance with a linear kernel also improved with the reduction of negative instances. This time, DCCA achieved better results than PCC with a 0.5% sample,

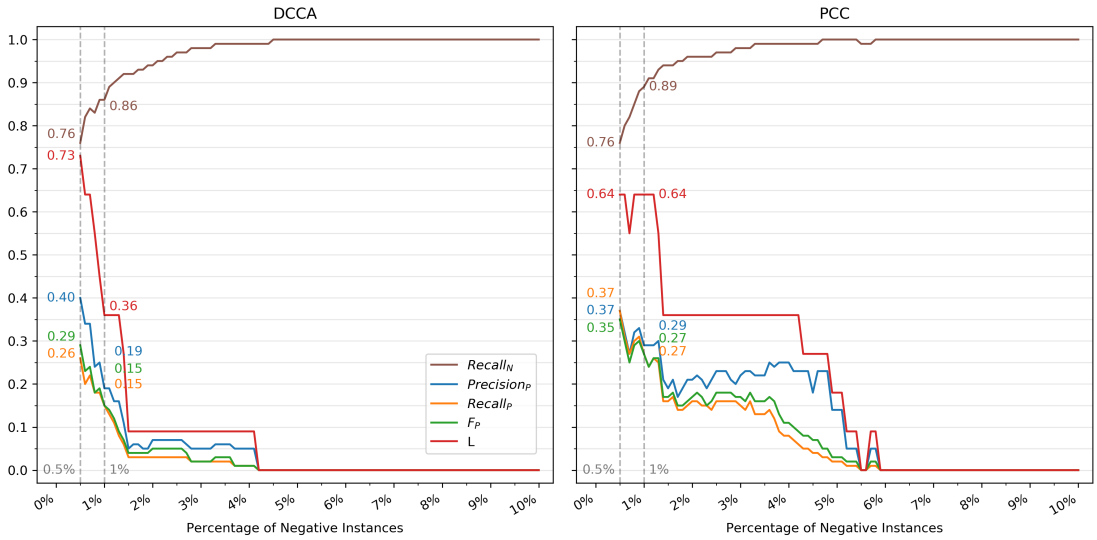

Figure 6.11: SVM's performance with a linear kernel along with the reduction of negative instances. Note that we used an 11-fold cross-validation on the training set and a time window of 120 minutes.

but its cross-validation results are less consistent than with a sample of 1%. Although PCC's results for a 0.5% sample also suffer from inconsistency, the results for other sample sizes are better. Note that the misclassification of negative instances also increased like in NB.

As Figure 6.12 shows and contrary to our other two classifiers, the SVM with an RBF kernel has less difficulty in detecting leakages with a larger sample. However, instead of learning the difference between classes, it seems to be mostly misclassifying negative instances to classify some positive ones. Regarding the correlation methods, although DCCA detects more leakages, PCC provides a more consistent behavior. Moreover, the results using a 1% sample are also more reliable than with a 0.5% sample.

Although an SVM with an RBF kernel obtained better results in a challenging scenario, NB and the SVM with a linear kernel performed better when in the presence of a more balanced dataset. Contrary to the results obtained for the synthetic setting in section 6.1, PCC results are better and more stable than DCCA's. One possible explanation is that DCCA could be more sensitive to variations, as shown in the heatmaps of section 5.2, which is an advantage in a controlled scenario since most changes are leakages. However, in a real unstable scenario, this sensitivity could mislead the classifiers into thinking that negative instances are positive. Lastly, a sample of 0.5% (153 instances) produced better results across our three classifiers, but it was not generic enough to be consistent in all iterations of cross-validation. Thus, a sample of 1% (306 instances) is a safer choice even though its results are not as good.

Regarding the next sections' experiments, we decided to use PCC as the correlation method. Moreover,

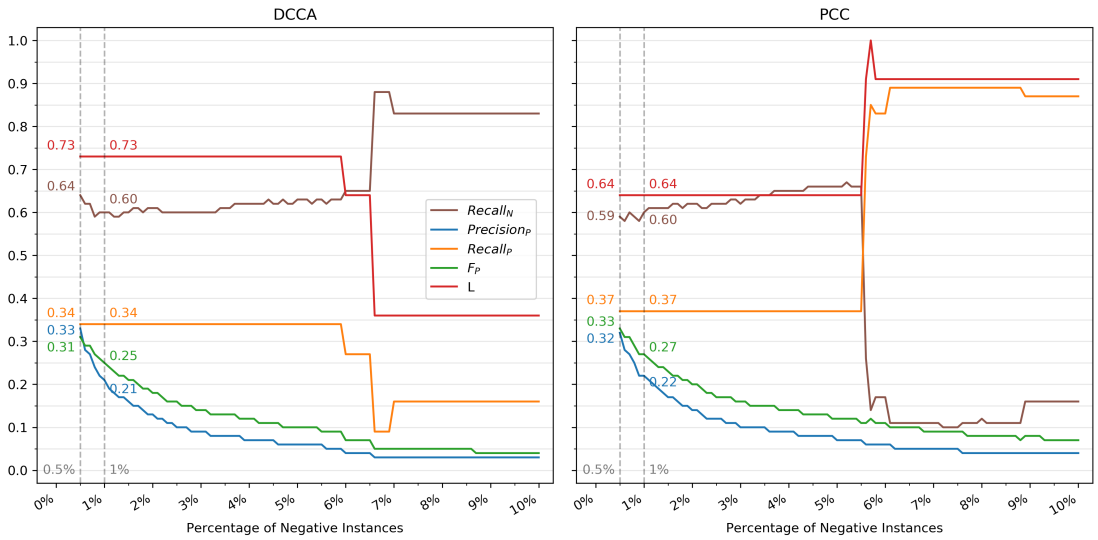

Figure 6.12: SVM's performance with a RBF kernel along with the reduction of negative instances. Note that we used an 11-fold cross-validation on the training set and a time window of 120 minutes.

we will compare the results obtained when training with a complete set versus one with a 1% sample of negative instances.

6.2.3 Feature Selection

Although this dataset does not have as many features as the synthetic one, it still could be useful to understand if any features are causing a performance decrease. Since the results obtained using an SVM with an RBF were the most promising when using all negative instances, we will start by analyzing how its performance varies along with the top k features, starting from the top 15 to the top 36.

As Figure 6.13 shows, the results differ a lot depending on the features. Additionally, the $recall_N$ is

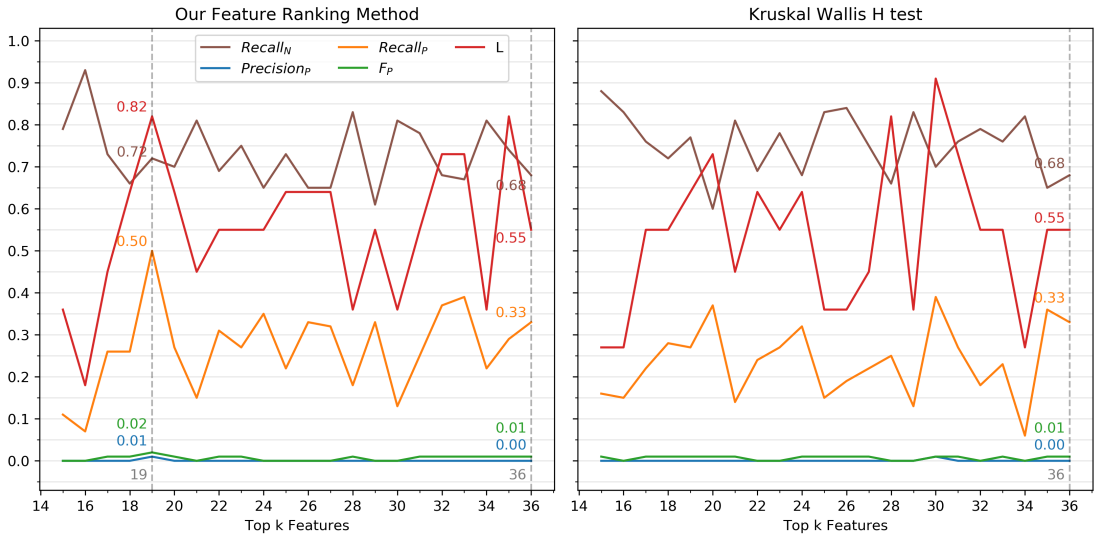

Figure 6.13: SVM's performance with an RBF kernel in the real setting along with the top k features selected by two feature ranking strategies. Note that we used an 11-fold cross-validation on the training set, PCC as the correlation method, and a time window of 120 minutes.

constantly dropping when the $recall_P$ increases, once again proving that the classifier can not properly differentiate the classes. Although this disparity does not allow us to be sure about which combination could be the best, the top 19 selected by our ranking method gets better results and is slightly more consistent than using all features. As shown in Figure 6.3, we also got better results for the synthetic setting with less than 30 features when using an RBF kernel.

Next, we will focus on the results obtained when using a 1% sample of random negative instances. Regarding NB, although its performance remains mostly stable, Figure 6.14 shows that the $recall_P$ improves slightly when the classifier uses more features.

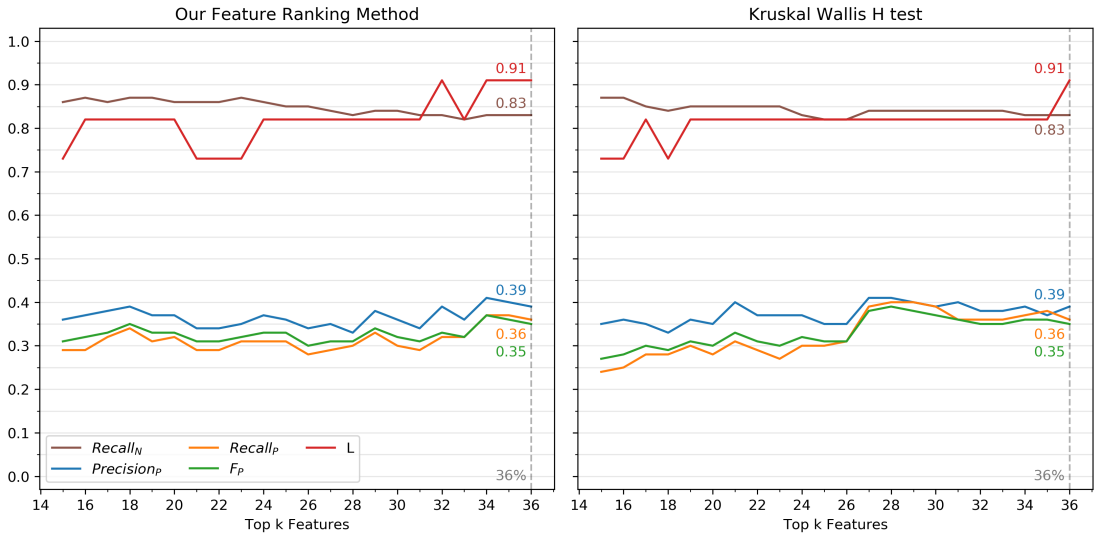

Figure 6.14: NB's performance on the real dataset (sample of 1% random negative instances) along with the top k features selected by two feature ranking strategies. Note that we used an 11-fold cross-validation on the training set, PCC as the correlation method, and a time window of 120 minutes.

As shown in Figure 6.15, the performance of the SVM with a linear kernel is not as stable as NB. Nonetheless, our feature ranking method selected a few feature combinations that obtained better results than if we used all 36 features. However, we believe that this difference is not significant enough to risk losing information when discarding others.

Regarding the SVM with an RBF kernel, as Figure 6.16 shows, we got very different results depending on the feature ranking method. With our method, we managed to detect more leakages and identify more positive instances. However, the $recall_N$ is also much lower than when we use all features. With the Kruskal–Wallis H test, the $recall_N$ improves with fewer features, but the $recall_P$ immediately drops.

Overall, with a 1% sample of negative instances, it seems that including all features is a safer choice. As detailed in section 6.1.2, note that in the synthetic setting, the performance of the classifiers improves considerably after reaching a minimum of 30 features. Since we only have 36, we believe that our classifiers would benefit from having more features/sensors. Lastly, although we managed to improve the performance when using all negative instances, the results obtained in each iteration of cross-validation still differ a lot more than with a 1% sample. Note that since feature selection did not lower the number of FP that the RBF kernel produces when using an 1% sample, we will not use it in the next sections.

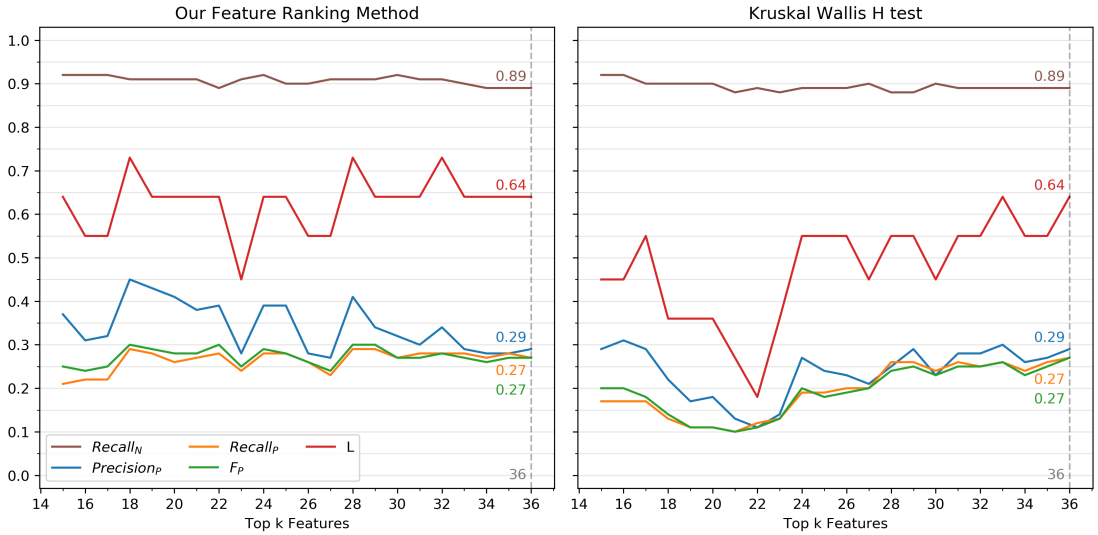

Figure 6.15: SVM's performance with a linear kernel on the real dataset (sample of 1% random negative instances) along with the top k features selected by two feature ranking strategies. Note that we used an 11-fold cross-validation on the training set, PCC as the correlation method, and a time window of 120 minutes.

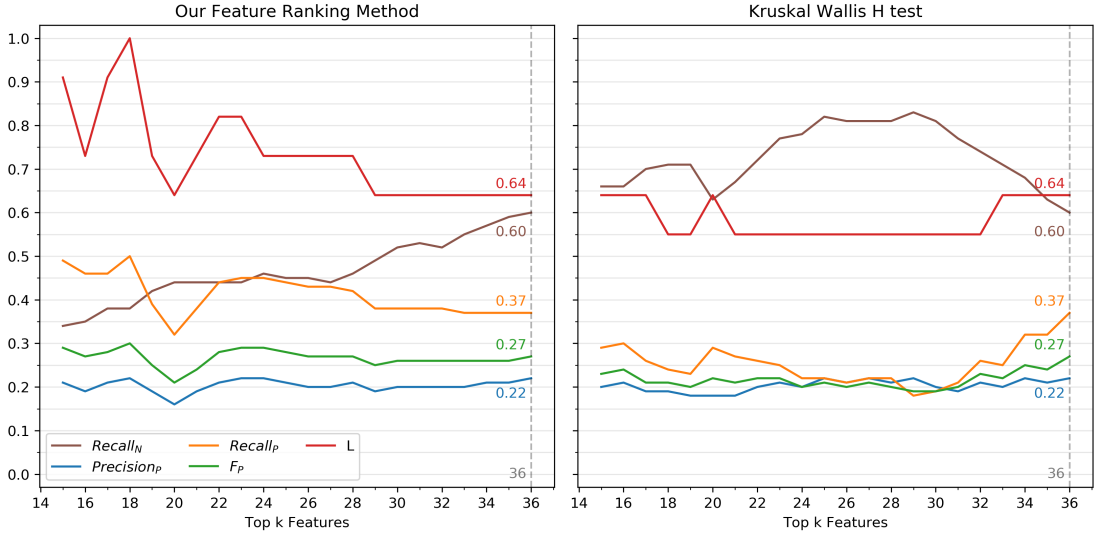

Figure 6.16: SVM's performance with an RBF kernel on the real dataset (sample of 1% random negative instances) along with the top k features selected by two feature ranking strategies. Note that we used an 11-fold cross-validation on the training set, PCC as the correlation method, and a time window of 120 minutes.

6.2.4 Time Window Size

As we have seen in section 6.1.3, the size of the time window affects the results obtained. Therefore, it is crucial to understand how different sizes affect the performance of our three classifiers in our real setting. Until now, we have been using a time window of 120 minutes, but we prepared three additional ones of 60, 180, and 240 minutes.

Table 6.6 presents the results obtained for the complete training set using an SVM with an RBF kernel. Note that, for the complete training set, we used the top 19 features selected by our feature ranking method. Regarding the results of the SVM, we can see that the classifier practically does not

Classifier	Correlation Method	TWS	#F	$Recall_N$	$Prec_P$	$Recall_P$	F_P	L
SVM (RBF Kernel)	PCC	60	19	0.79	0.01	0.25	0.01	0.91
		120	19	0.72	0.01	0.50	0.02	0.82
		180	19	0.88	0.00	0.01	0.00	0.18
		240	19	0.90	0.02	0.01	0.01	0.09

Table 6.6: SVM's performance with an RBF kernel on the complete real dataset using four different time window sizes. We used the top 19 features selected by our feature ranking method and an 11-fold cross-validation on the training set.

identify positive instances with time windows of 180 and 140 minutes. With 60 minutes, the SVM identifies one more leakage than with 120 minutes, but it only identifies 25% of positive instances. Therefore, a 120 minute window appears to be more reliable than the others.

Regarding the training set with a sample of 1% random negative instances, Table 6.7 presents the classifiers' results obtained with four different time windows. When using NB, we can see that the results

Classifier	Correlation Method	TWS	#F	$Recall_N$	$Prec_P$	$Recall_P$	F_P	L
Naive Bayes	PCC	60	36	0.74	0.36	0.43	0.37	0.91
		120	36	0.83	0.39	0.36	0.35	0.91
		180	36	0.83	0.34	0.39	0.36	0.64
		240	36	0.83	0.40	0.36	0.36	0.73
SVM (Linear Kernel)	PCC	60	36	0.88	0.25	0.11	0.16	0.55
		120	36	0.89	0.29	0.27	0.27	0.64
		180	36	0.87	0.20	0.20	0.19	0.45
		240	36	0.90	0.31	0.21	0.23	0.55

Table 6.7: Classifiers' performance on the real dataset (sample of 1% random negative instances) using four different time window sizes. Note that we used an 11-fold cross-validation on the training set.

for a time window of 120 and 240 are very similar, but the former allows NB to detect more leakages. However, it identifies fewer positive instances than a 60 minute window. Concerning the SVM with a linear kernel, a 120 minute window detects more leakages and identifies more positive instances without sacrificing the classification of negative instances.

Overall, a 60 and a 120 minute window provided better results than larger time windows. As we have seen in section 5.5, the correlation over time for larger windows looks more stable, possibly camouflaging the leakages. Since a 120 minute window obtained better results for the SVMs, we will keep using it in the next sections' experiments. Lastly, it is also important to acknowledge the size difference between the time windows from the synthetic and real settings. Note that the granularity of the synthetic dataset is only 10 minutes, whereas this one is 1 minute. So, although the synthetic time window is larger in time, it contains fewer points. For example, if we used a 60 minute window in the synthetic dataset, the methods would only have 6 points to calculate the correlation value. Therefore, the correlation methods need to use a larger time window in the synthetic setting to capture the true correlation value between time series. Consequentially, we also do not need to use a time window that large since the granularity of the real dataset is high enough.

6.2.5 Threshold Adjustment

The last step of hyperparameter tuning focuses on three key points, namely, (1) understanding what the best performance our classifiers can achieve is, (2) assessing if the default threshold is preventing our classifiers from achieving optimal results, and (3) finding the optimal threshold. It is important to note that in both NB's and SVM's case, the default threshold is zero.

Receiver Operating Characteristics Analysis

As mentioned in section 4.3.2, the Receiver Operating Characteristics (ROC) plot helps us understand what the best performance a classifier can achieve is by changing its default threshold. It is important to note that the ROC plot does not explicitly indicate how many leakages the classifiers detected.

We started by plotting the TPR and the FPR of the complete training set over all possible values of the threshold, resulting in the ROC curves shown in Figure 6.17. Note that the 11-fold cross-validation

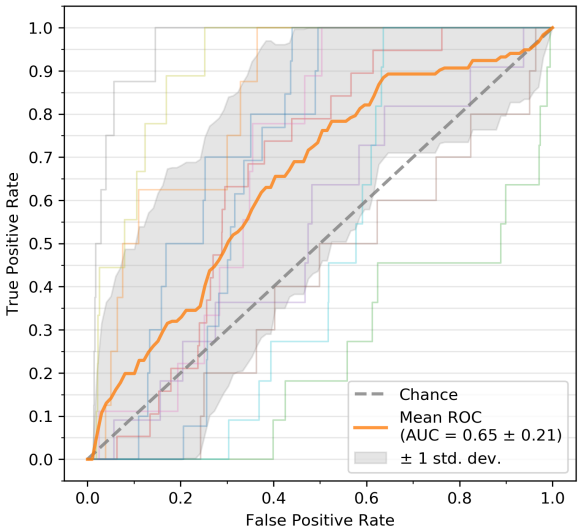

Figure 6.17: SVM's ROC plot of an 11-fold cross-validation with an RBF kernel on the complete real dataset. Note that we used the top 19 features selected by our feature ranking method, PCC as the correlation method, and a time window of 120 minutes.

produces 11 ROC curves, which differ a lot from one another. Additionally, some iterations of the SVM got results worse than random, i.e., below the chance line, which could we could solve by flipping the predicted classes. Unfortunately, that would only be possible if the other curves were also worst than random. Regarding the training set with a sample of 1% random negative instances, Figure 6.18 presents the classifiers' ROC plots for an 11-fold cross-validation. The results for the iterations of both classifiers not only vary a lot, but some were also worse than random. Ideally, the curves should be similar and closer to the upper left corner.

Overall, all iterations of our classifiers generated distinct ROC curves, reflecting the disparity of the results between iterations and proving that the dataset is not generic enough. Moreover, the three mean curves are very close to the chance line, confirming that the classifiers show a lot of difficulty in differentiating the positive instances from the negative ones. Lastly, the ROC curves showed that it is

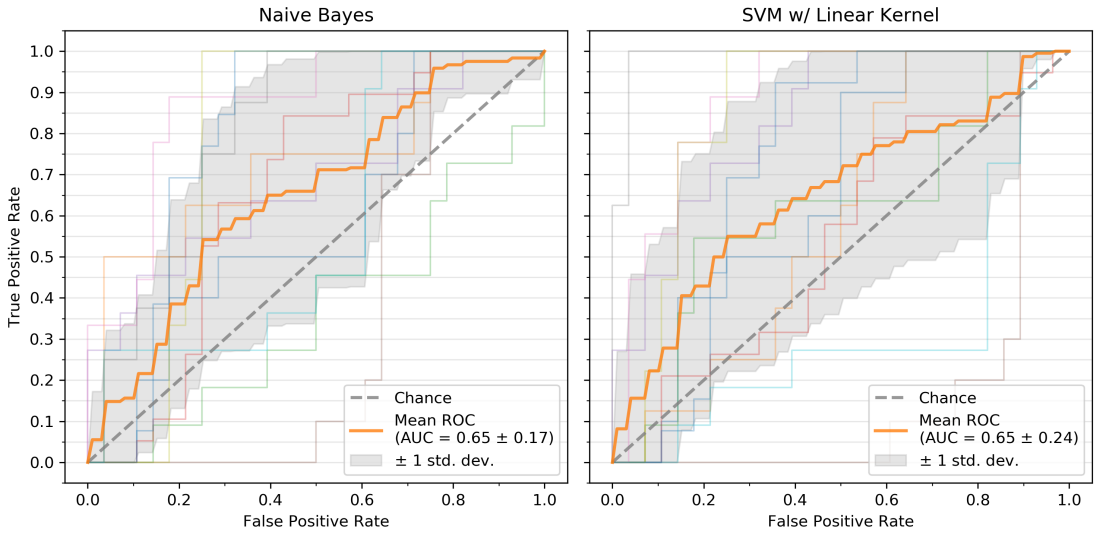

Figure 6.18: Classifiers' ROC plot of an 11-fold cross-validation on the real dataset (sample of 1% random negative instances). Note that we used PCC as the correlation method and a time window of 120 minutes.

not possible to improve the $recall_P$ without misclassifying a large number of negative instances.

Threshold Values

Although ROC plots allow us to quickly understand what the best performance a classifier can achieve is, it does take into consideration the number of leakages detected. Therefore, we plotted the performance measures along with possible threshold values, allowing us to understand their impact on our classifiers.

Regarding the complete training set, Figure 6.19 shows the impact of threshold variation on the SVM with an RBF kernel. Here we can clearly see that lowering the threshold increases the number of FP,

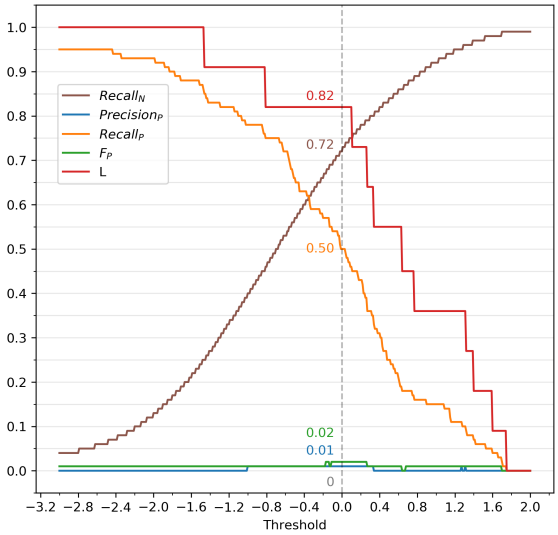

Figure 6.19: SVM's performance with an RBF kernel along with different thresholds on the complete real dataset. Note that we used the top 19 features selected by our feature ranking method, an 11-fold cross-validation on the training set, PCC as the correlation method, and a time window of 120 minutes.

confirming the results obtained in the ROC plot. Ideally, the $recall_P$ and L should be closer to the upper

right corner, and the $recall_N$ to the left one. Concerning the training set with a sample of 1% random negative instances, Figure 6.20 shows that if we lower the threshold until we reach $recall_N = 0.72$, we can identify more positive instances and detect more leakages than with the whole training set.

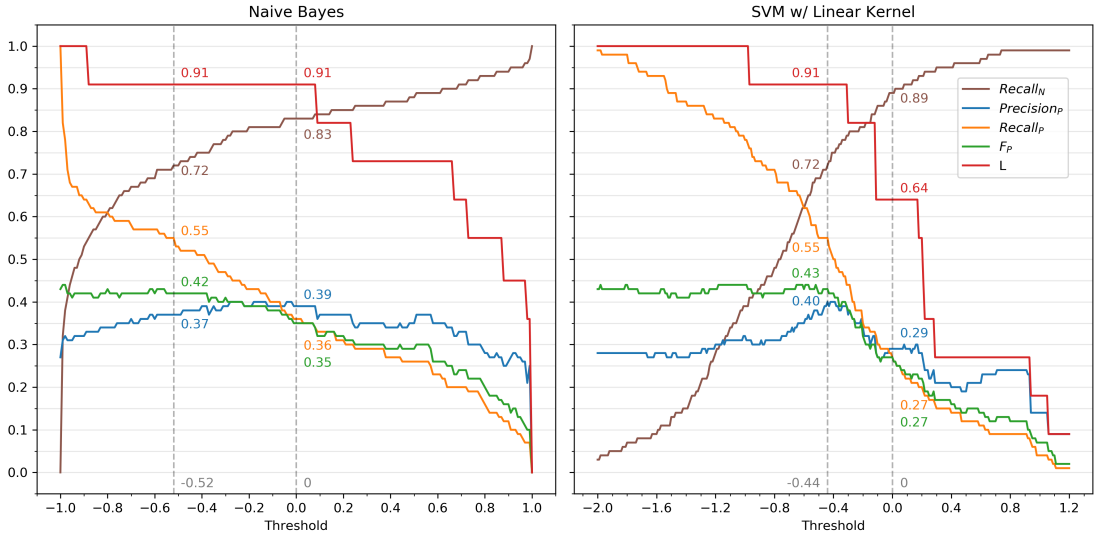

Figure 6.20: Classifiers' performance along with different thresholds on the real dataset (sample of 1% random negative instances). Note that we used an 11-fold cross-validation on the training set, PCC as the correlation method and a time window of 120 minutes.

Overall, if we were to chose a different threshold for NB and the SVM with a linear kernel, we would technically have a better performance than the SVM with an RBF kernel. However, since the iteration results are very distinct, we could be overfitting the data even more.

6.2.6 Test Set Results

In the previous sections, we used an 11-fold cross-validation on the training set to assess the performance of our classifiers and tune its hyperparameters. The mean results obtained after the hyperparameterization are summarized in Table 6.8. Although the cross-validation results were inconsistent, giving no guarantee

NI	Classifier	Correlation Method	TWS	#F	$Recall_N$	$Prec_P$	$Recall_P$	F_P	L
100%	SVM (RBF Kernel)	PCC	120	19	0.72	0.01	0.50	0.02	0.82
1%	Naive Bayes	PCC	120	36	0.83	0.39	0.36	0.36	0.91
	SVM (Linear Kernel)	PCC	120	36	0.89	0.29	0.27	0.27	0.64

Table 6.8: Results obtained by our classifiers using an 11-fold cross-validation on the training set. Regarding the SVM with an RBF kernel, we used the top 19 features selected by our feature ranking method. Lastly, note that NI refers to the percentage of negative instances used in the training set.

that the hyperparameters chosen are adequate, we will present the results obtained with the test set.

As previously described, the test set has a total of 9 positive instances from the same leakage but from different time windows. Like the others, we do not have information about its size and actual beginning, so we need to be careful about the conclusions we draw regarding the classification of these instances.

Table 6.9 shows the overall results with the test set for our three classifiers using the hyperparameters

defined earlier. As expected, the results obtained with the test set are different from the cross-validation

NI	Classifier	Correlation Method	TWS	#F	$Recall_N$	$Prec_P$	$Recall_P$	F_P	L
100%	SVM (RBF Kernel)	PCC	120	19	0.54	0.00	0.44	0.01	1
1%	Naive Bayes	PCC	120	36	0.57	0.00	0.22	0.00	1
	SVM (Linear Kernel)	PCC	120	36	0.59	0.00	0.11	0.00	1

Table 6.9: Results obtained by our classifiers on the real test set. Regarding the SVM with an RBF kernel, we used the top 19 features selected by our feature ranking method.

results. Although the $recall_N$ for the SVM with an RBF kernel is slightly lower than for other classifiers, its results are closer to the cross-validation ones. Additionally, we can see that a higher $recall_N$ results in a lower number of TP, meaning that the classifiers have to misclassify negative instances to identify the positive ones. Nonetheless, all classifiers identified at least one positive instance, so they all successfully detected the leakage.

Regarding the dataset imbalance, the results obtained with a 1% sample of negative instances are similar to the ones obtained with the SVM with an RBF kernel. However, it is important to note that both NB and the SVM with a linear kernel performed better with fewer negative instances. Therefore, we believe that reducing the negative instances is promising, but the inconsistency of the positive instances is preventing the classifiers from achieving good results.

Overall, the test set confirms that it is very difficult for classifiers to learn the difference between positive and negative instances in a real WDN. We identified several problems that may contribute to these results:

- *Low number of leakages* – Since we do not have many examples of what a leakage is, our classifiers can not generalize that information and end up overfitting the data;
- *Lack of information regarding the size and actual beginning of the leakage* – The results in section 5.3 show that it is easier to detect larger leakages, especially when the time window covers its beginning. For example, our leakages could be small, thus preventing our classifiers from learning and detecting them;
- *Location of leakages* – The leakages occur in different points of the network, affecting the correlations between downstream and upstream sensors, which hampers the generalization of leakage dynamics;
- *Poor sensor coverage* – Section 6.1.2 showed that the performance of the classifiers improved considerably with more than 30 features. Additionally, the synthetic dataset is also composed of data from sensors placed throughout the network. Therefore, we believe that our classifiers would benefit a lot from sensor expansion and relocation;
- *Network changes and interventions* – They can disrupt the natural behavior of the network and make the learning process more complicated.

Chapter 7

Visualization Tool

To facilitate the discovery of descriptors of network dynamics, we developed a visualization tool that supports the analysis of correlations between pairs of sensors in a WDN. This tool has two main components, namely, the analysis settings and the visualization, and was developed using the Bootstrap[1] CSS framework and Plotly[2] Javascript library.

The analysis settings component shown in figures 7.1 and 7.2 allows the users to choose (1) the time series they want to analyze and (2) the analysis settings, respectively. In the target time series section

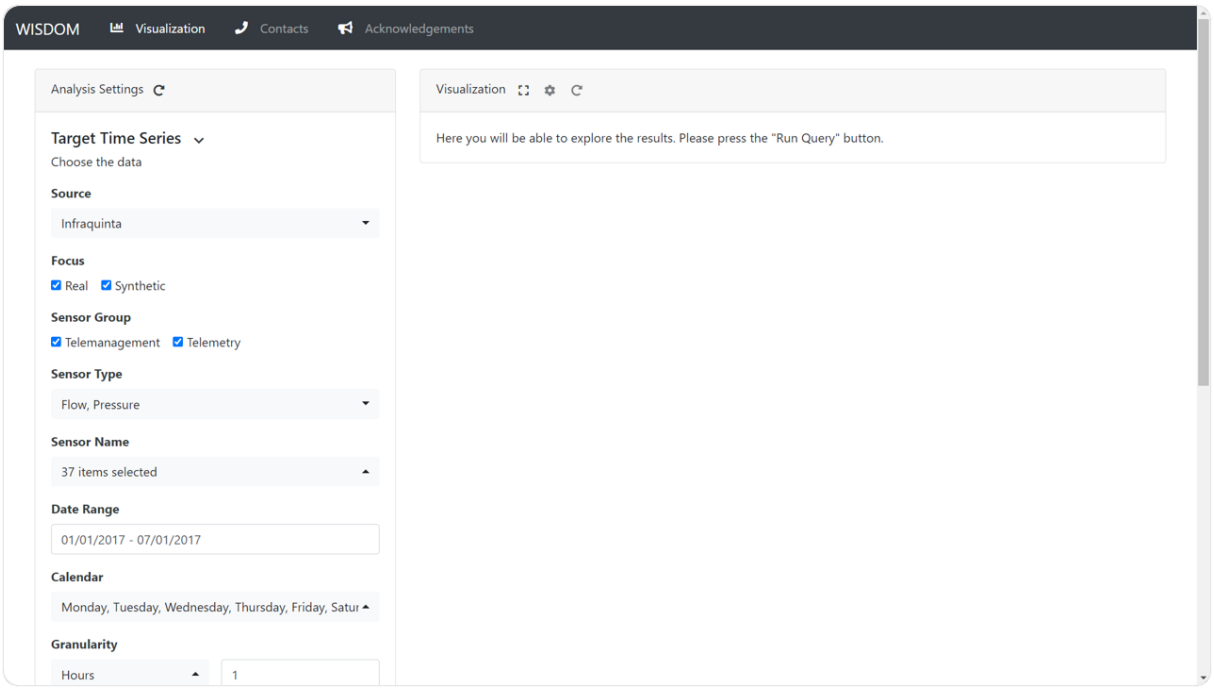

Figure 7.1: Target time series section of the analysis settings component of our visualization tool.

(Fig. 7.1), they can select the water distribution network, the sensors, the period of time, the weekdays to include, and the granularity of the time series. Then, in the data analysis section (Fig. 7.2), the users can choose to calculate the correlation for the full selected period of time or chunks of time. They can

[1] https://getbootstrap.com/
[2] https://plotly.com/javascript

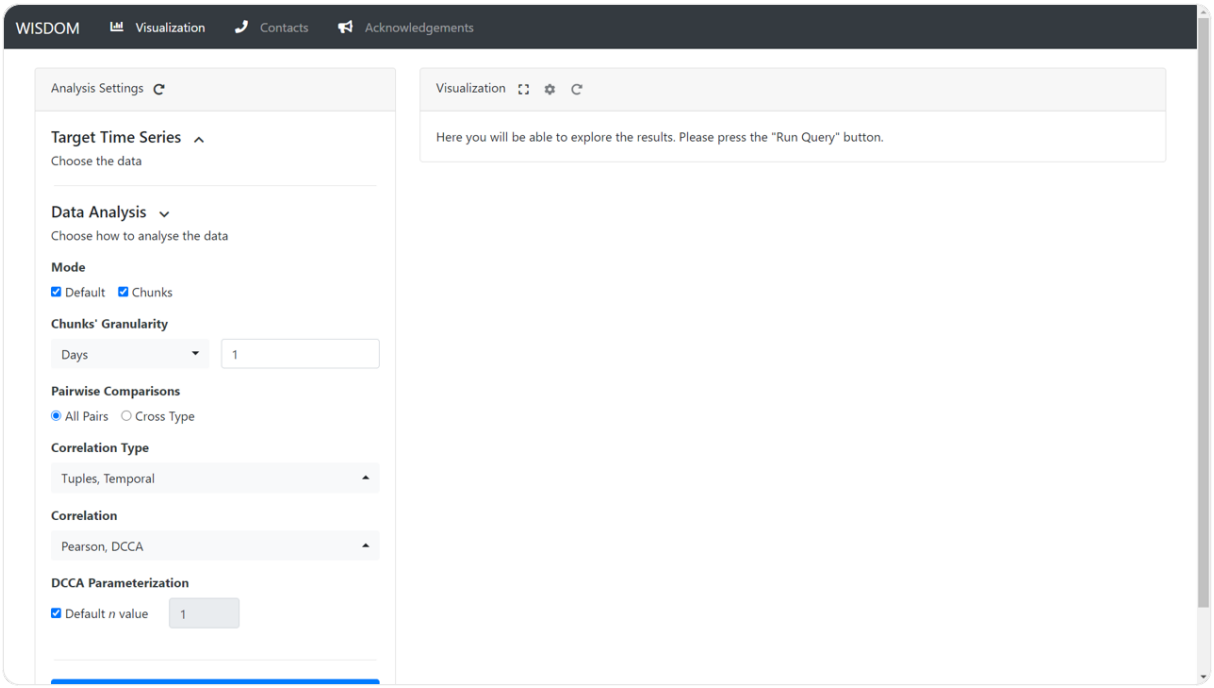

Figure 7.2: Data analysis section of the analysis settings component of our visualization tool.

also pick the correlation methods and choose the n parameter of DCCA.

Regarding the visualization component shown in figures 7.3 and 7.4, they allow the users to (1) visualize the time series and (2) explore the correlations between pairs of sensors. In the line chart

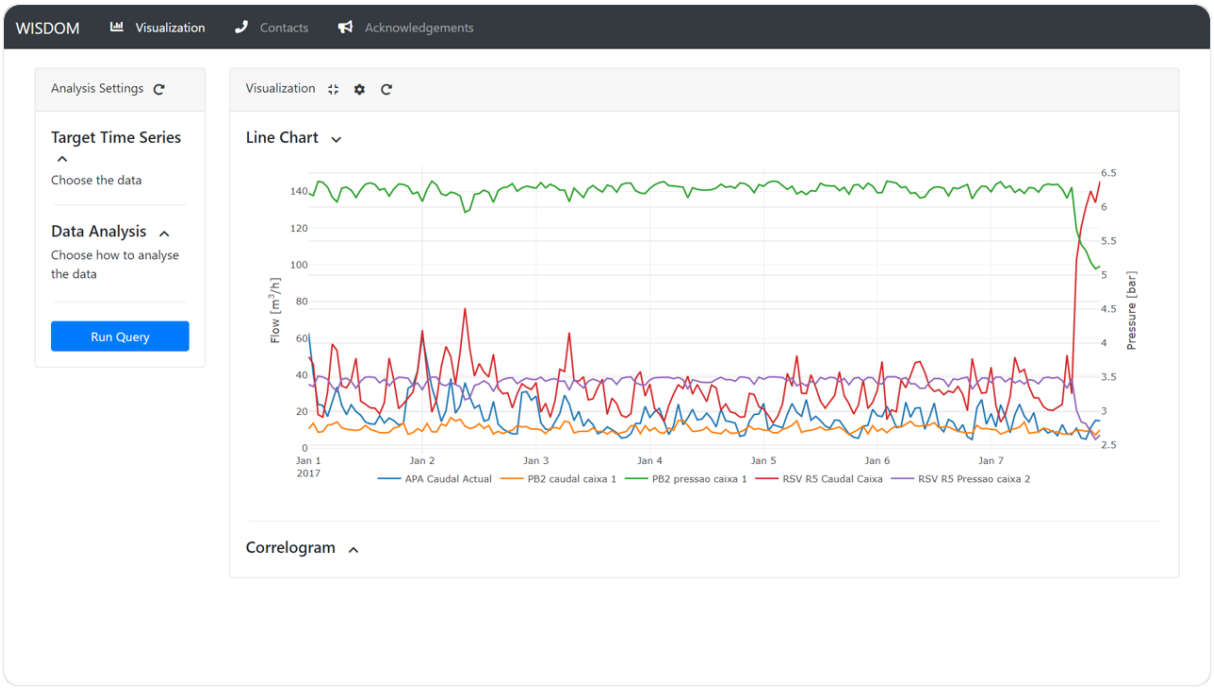

Figure 7.3: Line chart section of the visualization component of our visualization tool.

section (Fig. 7.3), the users can zoom in and select specific time series. Depending previous selection, the correlogram section (Fig. 7.4) allows the users to select the correlation method, order by correlation value and distance, and visualize the correlation for the full period of time or by chunks of time.

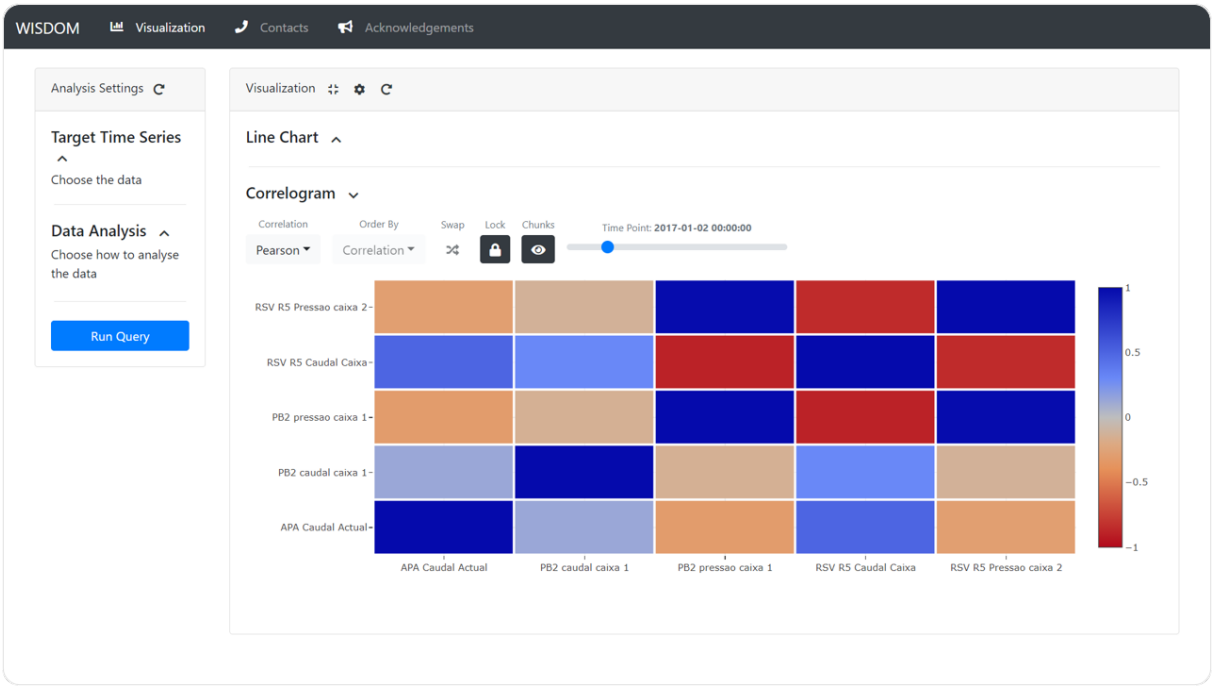

Figure 7.4: Correlogram section of the visualization component of our visualization tool.

Chapter 8

Conclusions

Our work explored and presented a solution to detect burst leakages in WDNs through the classification of time series data collected by WDN's sensors. The solution used the correlation values between pairs of sensors along time as descriptors of network dynamics for later predict the leakage dynamics, which can be easily generalized to other WDNs. Our experiments were conducted in a (1) synthetic setting, i.e., a controlled environment without noise and network changes, and in a (2) real challenging setting with highly irregular consumption patterns provided by Infraquinta's WDN.

Regarding the descriptors of dynamics, DCCA provided better results than PCC in the synthetic setting. However, in the real setting, PCC results were better and more stable than DCCA's. One possible explanation is that DCCA could be more sensitive to variations, which is an advantage in a controlled scenario since most changes are leakages. However, in a real unstable scenario, this sensitivity could mislead the classifiers into thinking that negative instances are leakages. Additionally, increasing the size of the time window does not always enhance the classifiers' performance and might camouflage the leakages. Specifically, a 60 and a 120 minute window provided better results than a 180 and a 240 minute window. Our experiments also showed that the maximum disruption of the correlation happens when the middle of the time window aligns with the beginning of the leakage. Moreover, a high number of features does not always result in the best performance but, in the synthetic setting, the classifiers needed a minimum of 30 features to achieve desirable results. For the real setting, since the dataset only has 36 features, the results indicated that including them all was the safer choice. Therefore, we believe that our classifiers would benefit a lot from sensor expansion and relocation. Lastly, both our feature relevance ranking method and Kruskal–Wallis H test provided good results depending on the setting.

Concerning the predictors of dynamics, the final results obtained for the real setting with a 1% sample of negative instances are similar to the ones obtained with all of them when using the SVM with an RBF kernel. However, the performance of NB and SVM with a linear kernel improved considerably after the reduction of these instances. Therefore, we believe that the reduction of negative instances is promising. Still, the irregularity of the positive ones prevented the inconsistency of the positive ones is preventing the classifiers from achieving desirable results. Additionally, the classifiers exhibit confidence in the classification of leakages with coefficients higher than 0.1 in the synthetic setting. When dealing with

smaller leakages, they have difficulty differentiating them from negative instances. Overall, the SVMs obtained better results than NB, but the latter proved to be a very competitive classifier in the real setting and benefited the most from dataset reduction.

Finally, the results of the real setting are more inconsistent and, thus, less reliable than the synthetic ones. Note that this inconsistency is expected since our leakages have different profiles and the cross-validation only has one leakage to validate per iteration. Overall, it is difficult for classifiers to learn the difference between positive and negative instances. We identified several problems that may contribute to these results, namely, (1) the low number of leakages to learn from and consequent class imbalance, (2) the lack of information regarding the size and actual beginning of the leakage, (3) the location of leakages, (4) poor sensor coverage, and (5) network changes and interventions that disrupt the natural behavior of the WDN. Regarding the lack of leakages, note that this problem would eventually disappear with time since the classifiers can be updated.

8.1 Future Work

We identified the following future directions:

- Explore our solution in other WDNs, namely, Municipality of Barreiro and Municipality of Beja;
- Develop a classifier that is better prepared to handle the nature of the gathered descriptors (correlation-based features) and their inherent spatiotemporal dependencies;
- Comprehensively analyze the correlation disruptions and predictive behavior to identify the location and size of leakages;
- Detect background leakages;
- Support the placement of new sensors;
- Support the detection of changes in the status of active hydraulic elements.

Bibliography

[1] EPA. Drinking water distribution systems, 2019. URL `https://www.epa.gov/dwsixyearreview/drinking-water-distribution-systems`. Accessed in: 18 Oct 2019.

[2] R. Clark, C. Stafford, and J. Goodrich. Water distribution systems: A spatial and cost evaluation. *Journal of Water Resources Planning and Management*, 108:243–256, 10 1982.

[3] E. J. Lee and K. J. Schwab. Deficiencies in drinking water distribution systems in developing countries. *Journal of water and health*, 3(2):109–127, 2005.

[4] A. Colombo and B. Karney. Energy and costs of leaky pipes: Toward comprehensive picture. *Journal of Water Resources Planning and Management*, 128:441–450, 11 2002. doi: 10.1061/(ASCE)0733-9496(2002)128:6(441).

[5] A. J. Whittle, M. Allen, A. Preis, and M. Iqbal. *Sensor networks for monitoring and control of water distribution systems*. International Society for Structural Health Monitoring of Intelligent, 2013.

[6] N. Carriço, M. Amado, D. Covas, J. Estima, J. Borbinha, J. Figueira, L. Monteiro, M. Barreira, R. Henriques, S. Fernandes, and S. Vinga. Water intelligence system data project, 2018. Active period: 2018-present. Lisbon, Portugal: Fundação para a Ciência e Tecnologia (FCT). Public information available in `https://www.fct.pt`.

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

[8] W. M. Persons. *Indices of Business Conditions: An Index of General Business Conditions*, volume 1. Harvard University Press, 1919.

[9] E. B. Dagum and S. Bianconcini. *Seasonal adjustment methods and real time trend-cycle estimation*. Springer, 2016.

[10] M. J. Zaki, W. Meira Jr, and W. Meira. *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.

[11] A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 31(3):606–660, 2017.

[12] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3–24, 2007.

[13] I. Rish et al. An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46, 2001.

[14] M. A. Hearst, S. T. Dumais, E. Osuna, J. Platt, and B. Scholkopf. Support vector machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998.

[15] M. Hofmann. Support vector machines-kernels and the kernel trick. *Notes*, 26(3), 2006.

[16] G. Cembrano, G. Wells, J. Quevedo, R. Pérez, and R. Argelaguet. Optimal control of a water distribution network in a supervisory control system. *Control engineering practice*, 8(10):1177–1188, 2000.

[17] T. M. Walski, D. V. Chase, D. A. Savic, W. Grayman, S. Beckwith, and E. Koelle. Advanced water distribution modeling and management. *Civil and Environmental Engineering and Engineering Mechanics Faculty Publications*, 2003.

[18] D. F. Young, B. R. Munson, T. H. Okiishi, and W. W. Huebsch. *A brief introduction to fluid mechanics*. John Wiley & Sons, 2010.

[19] M. Farley, S. Water, W. Supply, S. C. Council, W. H. Organization, et al. Leakage management and control: A best practice training manual. Technical report, Geneva: World Health Organization, 2001.

[20] M. Farley and S. Trow. *Losses in water distribution networks*. IWA publishing, 2003.

[21] Ofwat. Leakage, 2019. URL `https://www.ofwat.gov.uk/households/supply-and-standards/leakage`. Accessed in: 9 Oct 2019.

[22] L. Araujo, H. Ramos, and S. Coelho. Pressure control for leakage minimisation in water distribution systems management. *Water resources management*, 20(1):133–149, 2006.

[23] EPAL. Tele-management, 2019. URL `https://www.epal.pt/EPAL/en/menu/our-water/supply-system/tele-management`. Accessed in: 9 Oct 2019.

[24] A. Bargiela and G. D. Hainsworth. Pressure and flow uncertainty in water systems. *Journal of Water Resources Planning and Management*, 115(2):212–229, 1989.

[25] Merriam-Webster, 2019. URL `https://www.merriam-webster.com`. Accessed in: 21 Oct 2019.

[26] L. A. Rossman et al. *EPANET 2: users manual*. US Environmental Protection Agency. Office of Research and Development, 2000.

[27] N. P. Sonaje and M. G. Joshi. A review of modeling and application of water distribution networks (wdn) softwares. *International Journal of Techical Research and Applications*, 3(5):174–178, 2015.

[28] S. García, J. Luengo, and F. Herrera. *Data preprocessing in data mining*. Springer, 2015.

[29] M. Lepot, J.-B. Aubin, and F. H. Clemens. Interpolation in time series: An introductive overview of existing methods, their performance criteria and uncertainty assessment. *Water*, 9(10):796, 2017.

[30] D. Kondrashov and M. Ghil. Spatio-temporal filling of missing points in geophysical data sets. *Nonlinear Processes in Geophysics*, 13(2):151–159, 2006.

[31] M. Ghil, M. Allen, M. Dettinger, K. Ide, D. Kondrashov, M. Mann, A. W. Robertson, A. Saunders, Y. Tian, F. Varadi, et al. Advanced spectral methods for climatic time series. *Reviews of geophysics*, 40(1):3–1, 2002.

[32] J. P. Musial, M. M. Verstraete, and N. Gobron. Comparing the effectiveness of recent algorithms to fill and smooth incomplete and noisy time series. *Atmospheric chemistry and physics*, 11(15): 7905–7923, 2011.

[33] R. Vautard, P. Yiou, and M. Ghil. Singular-spectrum analysis: A toolkit for short, noisy chaotic signals. *Physica D: Nonlinear Phenomena*, 58(1-4):95–126, 1992.

[34] K. Hocke and N. Kämpfer. Gap filling and noise reduction of unevenly sampled data by means of the lomb-scargle periodogram. *Atmospheric Chemistry and Physics*, 9(12):4197–4206, 2009.

[35] S. Kandasamy, F. Baret, A. Verger, P. Neveux, and M. Weiss. A comparison of methods for smoothing and gap filling time series of remote sensing observations–application to modis lai products. *Biogeosciences*, 10(6):4055–4071, 2013.

[36] P. H. Eilers. A perfect smoother. *Analytical chemistry*, 75(14):3631–3636, 2003.

[37] E. T. Whittaker. On a new method of graduation. *Proceedings of the Edinburgh Mathematical Society*, 41:63–75, 1922.

[38] F. R. Macaulay et al. The smoothing of time series. *NBER Books*, 1931.

[39] R. J. Hyndman and G. Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[40] J. Shiskin. *The X-11 variant of the census method II seasonal adjustment program*. US Government Printing Office, 1965.

[41] G. P. Zhang and M. Qi. Neural network forecasting for seasonal and trend time series. *European journal of operational research*, 160(2):501–514, 2005.

[42] V. Gomez and A. Maravall. Programs tramo (time series regression with arima noise, missing observations, and outliers) and seats (signal extraction in arima time series). instructions for the user. *Documento de Trabajo*, 9628, 1996.

[43] S. Bisgaard and M. Kulahci. *Time series analysis and forecasting by example*. John Wiley & Sons, 2011.

[44] B. Fischer. *Decomposition of time series: comparing different methods in theory and practice.* Eurostat, 1995.

[45] C. C. Hood. Comparison of time series characteristics for seasonal adjustments from seats and x-12-arima. *ASA proceedings, business and economic statistics section, alexandria, VA: ASA*, 2002.

[46] R. B. Cleveland, W. S. Cleveland, J. E. McRae, and I. Terpenning. Stl: a seasonal-trend decomposition. *Journal of official statistics*, 6(1):3–73, 1990.

[47] A. Boggess and F. J. Narcowich. *A first course in wavelets with Fourier analysis.* John Wiley & Sons, 2015.

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

[49] J. Serra and J. L. Arcos. An empirical evaluation of similarity measures for time series classification. *Knowledge-Based Systems*, 67:305–314, 2014.

[50] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

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

[52] P. Senin. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, 855(1-23):40, 2008.

[53] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and information systems*, 7(3):358–386, 2005.

[54] E. J. Keogh and M. J. Pazzani. Derivative dynamic time warping. In *Proceedings of the 2001 SIAM international conference on data mining*, pages 1–11. SIAM, 2001.

[55] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 285–289. ACM, 2000.

[56] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[57] L. Chen, M. T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM, 2005.

[58] P.-F. Marteau. Time warp edit distance with stiffness adjustment for time series matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):306–318, 2008.

[59] S. Lhermitte, J. Verbesselt, W. W. Verstraeten, and P. Coppin. A comparison of time series similarity measures for classification and change detection of ecosystem dynamics. *Remote sensing of environment*, 115(12):3129–3152, 2011.

[60] M. Bermudez-Edo, P. Barnaghi, and K. Moessner. Analysing real world data streams with spatio-temporal correlations: Entropy vs. pearson correlation. *Automation in Construction*, 88:87–100, 2018.

[61] J. C. de Winter, S. D. Gosling, and J. Potter. Comparing the pearson and spearman correlation coefficients across distributions and sample sizes: A tutorial using simulations and empirical data. *Psychological methods*, 21(3):273, 2016.

[62] Y.-W. Laih. Measuring rank correlation coefficients between financial time series: A garch-copula based sequence alignment algorithm. *European Journal of Operational Research*, 232(2):375–382, 2014.

[63] B. Podobnik and H. E. Stanley. Detrended cross-correlation analysis: a new method for analyzing two nonstationary time series. *Physical review letters*, 100(8):084102, 2008.

[64] D. Horvatic, H. E. Stanley, and B. Podobnik. Detrended cross-correlation analysis for non-stationary time series with periodic trends. *EPL (Europhysics Letters)*, 94(1):18007, 2011.

[65] G. F. Zebende. Dcca cross-correlation coefficient: quantifying level of cross-correlation. *Physica A: Statistical Mechanics and its Applications*, 390(4):614–618, 2011.

[66] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh. *Feature extraction: foundations and applications*, volume 207. Springer, 2008.

[67] M. Ringnér. What is principal component analysis? *Nature biotechnology*, 26(3):303–304, 2008.

[68] L. McInnes, J. Healy, and J. Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[69] G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014.

[70] G. Heiman. *Basic Statistics for the Behavioral Sciences*. Cengage Learning, 2013.

[71] P. E. McKight and J. Najab. Kruskal-wallis test. *The corsini encyclopedia of psychology*, pages 1–1, 2010.

[72] R. Li, H. Huang, K. Xin, and T. Tao. A review of methods for burst/leakage detection and location in water distribution systems. *Water Science and Technology: Water Supply*, 15(3):429–441, 2015.

[73] R. Puust, Z. Kapelan, D. Savic, and T. Koppel. A review of methods for leakage management in pipe networks. *Urban Water Journal*, 7:25–45, 02 2010. doi: 10.1080/15730621003610878.

[74] K. Aksela, M. Aksela, and R. Vahala. Leakage detection in a real distribution network using a som. *Urban Water Journal*, 6(4):279–289, 2009.

[75] S. Valizadeh, B. Moshiri, and K. Salahshoor. Leak detection in transportation pipelines using feature extraction and knn classification. In *Pipelines 2009: Infrastructure's Hidden Assets*, pages 580–589. American Society of Civil Engineers, 2009.

[76] K. Adedeji, Y. Hamam, B. Abe, and A. Abu-Mahfouz. Leakage detection and estimation algorithm for loss reduction in water piping networks. *Water*, 9(10):773, 2017.

[77] M. Casillas, V. Puig, L. Garza-Castañón, and A. Rosich. Optimal sensor placement for leak location in water distribution networks using genetic algorithms. *Sensors*, 13(11):14984–15005, 2013.

[78] W. E. Hart and R. Murray. Review of sensor placement strategies for contamination warning systems in drinking water distribution systems. *Journal of Water Resources Planning and Management*, 136 (6):611–619, 2010.

[79] R. Pérez, V. Puig, J. Pascual, A. Peralta, E. Landeros, and L. Jordanas. Pressure sensor distribution for leak detection in barcelona water distribution network. *Water Science & Technology: Water Supply*, 9, 12 2009. doi: 10.2166/ws.2009.372.

[80] R. Sarrate, J. Blesa, F. Nejjari, and J. Quevedo. Sensor placement for leak detection and location in water distribution networks. *Water Science and Technology: Water Supply*, 14(5):795–803, 2014.

[81] M. Stephens, J. Vitkovsky, M. Lambert, A. Simpson, B. Karney, and J. Nixon. Transient analysis to assess valve status and topology in pipe networks. In *9 th International Conference on Pressure Surges*, 2004.

[82] J. Deuerlein, A. R. Simpson, and E. Gross. The never ending story of modeling control-devices in hydraulic systems analysis. In *Water Distribution Systems Analysis 2008*, pages 1–12. American Society of Civil Engineers, 2008.

[83] O. Piller and J. E. van Zyl. Modeling control valves in water distribution systems using a continuous state formulation. *Journal of Hydraulic Engineering*, 140(11):04014052, 2014.

[84] O. Giustolisi, Z. Kapelan, and D. Savic. Algorithm for automatic detection of topological changes in water distribution networks. *Journal of Hydraulic Engineering*, 134(4):435–446, 2008.

[85] O. Giustolisi and D. Savic. Identification of segments and optimal isolation valve system design in water distribution networks. *Urban Water Journal*, 7(1):1–15, 2010.

[86] T. Munzner. *Visualization analysis and design*. CRC press, 2014.