Faculty of Sciences and Tecnology

Department of Informatics Engineering

Time Series Analysis Framework

Pedro Moreira Costa

Dissertation in the context of the Master in Informatics Engineering, Specialization in Intelligent Systems advised by Prof. Nuno Lourenço and Eng. Miguel Oliveira presented to the Faculty of Sciences and Technology / Department of Informatics Engineering

July 2019

1 2 9 0

UNIVERSIDADE Ð COIMBRA

This page is intentionally left blank.

Abstract

A time series consists of a data structure that relates an event observation with a time instance. It is the type of data that occurs in a variety of fields and whose analysis (such as Forecasting and Anomaly Detection) provides a more sensitive understanding regarding events' behavior. Time series analysis is usually carried out through the use of plots and classical models. However, Machine Learning (ML) approaches have seen a rise in the state of art for Forecasting and Anomaly Detection because they provide comparable results at appropriate time and data constraints. However, one of their major drawbacks is how costly it is to obtain the best data preparation, model selection and parametrization.

With this in mind, this work's main goal is overcoming the complexity involved with data processing, model selection and model tuning, through the support of an autonomous approach selector, based on Case-based Reasoning (CBR) and Bayesian Optimization, present in both forecasting and anomaly detection frameworks. The framework handles Univariate Time Series (UTS) for Forecasting and Multivariate Time Series (MTS) for Forecasting and Anomaly Detection of different categories and bearing different attributes to deliver an optimized approach.

The drawn results for each type of analysis, for each type of time series, suffer from insufficient examples, especially MTS, but, for Forecasting UTS, the model selection results show an averaged Macro weighted F1-score of 0.38. Regarding the effects of Bayesian Optimization, all models produced results within an acceptable error (Symmetric Mean Absolute Percentage Error (sMAPE) lower than 16% and F1-score higher than 0.60), proving the efficacy of this component.

Keywords: Time Series, Forecasting, Anomaly Detection, Auto-ML, Case-Based Reasoning, Bayesian Optimization, Machine Learning, Deep Learning

This page is intentionally left blank.

Resumo

Uma série temporal é uma estrutura de dados que relaciona uma observação de um evento com um instante de tempo e o tipo de dados que é utilizado numa variedade de áreas, cuja análise providencia uma compreensão mais sensata em relação ao comportamento de um evento. É comum a análise de séries temporais ser desempenhada com recurso a vusializações gráficas e modelos clássicos. No entanto, os modelos de ML têm recebido atenção no estado da arte, no que diz respeito a previsão e deteção de anomalias, uma vez que apresentam resultados comparáveis, tendo em conta restrições de tempo e de dados. No entanto, uma das maiores desvantagens destes métodos é o custo associado a obter a melhor configuração de pré-processamento de dados, seleção do modelo e a sua parametrização.

Deste modo, o objectivo principal deste projeto trata por reduzir a complexidade associada à escolha do tipo de processamento de dados, seleção de modelo e a sua parametrização, através de um seletor automático de abordagens, baseado em Raciocínio Baseado em Casos e Otimização Bayesiana, tanto para uma framework de previsão, como uma de deteção de anomalias. A framework deve gerir métodos de previsão e deteção de anomalias para séries temporais uni e multivariadas, de diferentes categorias e constituídas por diferentes atributos para fornecer a abordagem ótima. Os resultados recolhidos para cada tipo de análise, por cada tipo de série temporal, foram comprometidos por insuficiência de exemplos, especialmente nos casos de séries temporais multivariadas. No entanto, para séries temporais univariadas, os resultados são mais razoáveis. O impacto de Otimização Bayesiana mostra que todos os modelos aos quais foi aplicada apresentavam resultados com um erro aceitável, demonstrando-se uma mais-valia.

Keywords: Séries Temporais, Previsão, Deteção de Anomalia, Aprendizagem de Máquina Automática, Aprendizagem de Máquina, Aprendizagem Profunda

This page is intentionally left blank.

Acknowledgements

The work described in this document expresses the pinnacle of my Masters in Informatics Engineering: a journey which was supported by many people in my life.

To my parents, who supported all my endeavors and made it possible for me to attend university.

To my advisor, Professor Nuno Lourenço, for all the insight associated with this project and for generally tranquilizing me at peak decision situations. To my friends at Department of Informatics Engineering (DEI) for enriching my time at Coimbra. To my hometown friends for keeping me company during the university off-times.

To my company advisor, Engineer Miguel Oliveira, for all the overview on the internship tasks, correct postures and healthy lifestyle choices, and assuming a position where we both evolved, in the area of Intelligent Systems. To my co-worker, Gisèle Pereira, for also enduring hard technical decisions and always pushing for this work, and me, to achieve its full potential.

To my friend and roommate, João Alves, for sharing course experiences, good and bad. To my friend, Inês Valentim, for being a constant example, since the Bachelor course, for integrity, workmanship and excellence.

To all of these people, a heartfelt thank you.

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

AI Artificial Intelligence. 35

AIC Akaike information criterion. 12

API Application Programming Interface. 46, 92

AR Autoregressive. 7, 8

ARIMA Autoregressive Integrated Moving Average. 6, 8, 9, 10, 11, 12, 49

ARMA Autoregressive Moving Average. 7, 8

ARMQE Average Relative Mean Quadratic Error. 10

AWS Amazon Web Services. 46

BGMM Bayesian Gaussian Mixture Model. xviii, 69, 84

BN Bayesian Network. 28

BNN Bayesian Neural Network. 12, 13, 17, 19, 25, 91

BSD Berkeley Software Distribution. 45

CART Classification and Regression Trees. xviii, 16, 19, 68, 79, 80, 81, 82

CBR Case-based Reasoning. iii, xv, xvi, xviii, xix, 2, 35, 55, 56, 57, 60, 61, 62, 63, 64, 65, 70, 72, 73, 74, 76, 77, 78, 79, 80, 81, 82, 83, 84, 90, 91

CNN Convolutional Neural Network. 45, 46

CNTK Microsoft Cognitive Toolkit. 46

CPU Central Processing Unit. 46

DA Direction Accuracy. 12, 17

DEI Department of Informatics Engineering. vii, 1, 53, 86

EI Expected Improvement. 36

EML Extreme Learning Machine. 15, 17

FAR False Alarm Rate. 24

FCTUC Faculty of Sciences and Technology of the University of Coimbra. 1

FFNN Feedforward Neural Network. 10, 20

FN False Negative. xviii, 60

FP False Positive. xviii, 60

GMM Gaussian Mixture Model. xviii, 69, 84

GP Gaussian Process. xv, xvi, 15, 19, 37, 68, 71, 73, 78, 79, 80, 81, 82, 91

GPU Graphics Processing Unit. 46

GRNN Generalized Regression Neural Network. xv, xviii, 18, 19, 66, 68, 79, 80

HMM Hidden Markov Model. 23, 24

HW Holt-Winters Exponential Smoothing. 6, 9, 10, 12, 17

IDS Intrusion Detection System. 27, 33

k-NN k-Nearest Neighbours. xv, xviii, 15, 17, 19, 45, 56, 57, 60, 62, 63, 66, 68, 70, 76, 77, 79, 81, 82, 84, 91

LSTM Long Short Term Memory. xv, xviii, 15, 20, 21, 22, 23, 25, 27, 37, 38, 46, 49, 66, 67, 69, 72, 78, 79, 84, 91

MA Moving Average. 7, 8

MAE Mean Absolute Error. 12, 17, 62

MAPE Mean Absolute Percentage Error. 9, 10, 12, 17

M-Competition Makridakis Competition. 1, 6, 44, 58, 77, 78, 90, 91

MEI Master in Informatics Engineering. 1

MIT Massachusetts Institute of Technology. 46

ML Machine Learning. iii, v, xviii, 2, 12, 13, 15, 17, 25, 40, 41, 42, 43, 44, 48, 50, 53, 65, 90

MLP Multi Layer Perceptron. xv, xviii, 11, 12, 13, 14, 15, 17, 19, 22, 25, 66, 67, 68, 70, 72, 78, 79, 84, 91

MSE Mean Squared Error. 10, 17, 22, 61, 62

MTS Multivariate Time Series. iii, xv, xvi, xix, 3, 49, 52, 56, 57, 58, 59, 60, 62, 63, 64, 65, 66, 73, 74, 76, 77, 80, 81, 82, 83, 90, 91

NDM Network Data Mining. 33

NN Neural Network. 1, 11, 14, 20, 22, 67, 91

OCSVM One-Class Support Vector Machine. 26, 27

PCA Principal Component Analysis. xv, 29, 30

RBFNN Radial Basis Function Neural Network. xv, 13, 14, 15, 18, 19

REST Representational State Transfer. 92

RMSE Root Mean Square Error. 12, 14, 17, 62

RNN Recurrent Neural Network. xv, 19, 20, 21, 22, 23, 27, 45, 46

RSVM Robust Support Vector Machine. 27

sMAPE Symmetric Mean Absolute Percentage Error. iii, xviii, xix, 12, 13, 15, 16, 17, 19, 37, 38, 62, 72, 79, 80, 82, 91

SMETS Semi Metric Ensemble Time Series. xv, 56, 57, 60, 63, 64, 65, 76, 82, 90, 91

SMO Sequential Minimal Optimization. 12, 17, 27

SVM Support Vector Machine. xvi, xviii, 1, 16, 17, 26, 27, 33, 42, 45, 69, 70, 74, 83, 84

SVR Support Vector Regression. xvi, 15, 17, 25, 66, 72, 78, 79

TN True Negative. xviii, 60

TP True Positive. xviii, 60

TPR True Positive Rate. 24

TSFRESH Time Series Feature extraction based on Scalable Hypothesis Tests. 57, 62

UCI University of California Irvine. 44, 51, 58, 59, 90

UTS Univariate Time Series. iii, xv, xvi, xviii, 3, 49, 52, 56, 57, 58, 60, 62, 63, 64, 65, 66, 68, 72, 76, 77, 78, 79, 80, 82, 90, 91

WEKA Waikato Environment for Knowledge Analysis. 45

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

The following report presents the work developed in the lective year of 2018/2019 in the context of the curricular unit "Dissertation/Internship" from the Master in Informatics Engineering (MEI) with Specialization in Intelligent Systems from the Faculty of Sciences and Technology of the University of Coimbra (FCTUC).

The internship took place at the recently created Center for Cognitive Computing of Novabase[1]. Novabase is a company created in 1989 that specializes in a set of business areas (Financial Services, Energy, Transportation and Government). The Center for Cognitive Computing is dedicated to applying the latest cognitive technologies to deliver software solutions. Internship supervision was handled by **Eng. Miguel Oliveira** (Novabase, Cognitive Computing) and **Dr. Nuno Lourenço** (Department of Informatics Engineering (DEI), FCTUC).

1.1 Context

The analysis of events that occur along time is frequent in a variety of areas, ranging from marketing, to finance and engineering, while also applicable to IoT data and meteorological readings. Knowing how particular phenomena behave, and being able to predict their evolution, helps to identify problems, anticipate measures, and ultimately condition better decisions. While it entails many specifications, two common areas to time series analysis are the prediction of occurrences based on past behavior (forecasting) and the detection of irregular observations, in relation to the generality of the data instances (anomaly detection). These types of analysis are usually carried out with statistical approaches, but other techniques can also be considered, such as Neural Network (NN), Support Vector Machine (SVM) and others. The latter techniques' quality is tied to their configuration, which is characteristic of the data being fitted to them. Thus, a domain expert is crucial to select the most appropriate approach and to bridge data properties to hyperparameter tuning.

1.2 Motivation

Time series analysis has been carried out mostly by classical approaches, but recently also by machine learning ones. In either case, analysis relies heavily on the model choice and corresponding parameter tuning in order to adapt to different time series.

Although classical approaches are well established throughout the sectors where they are applied, machine learning approaches show promise, as:

- the results are almost comparable to pure statistical approaches, as shown through popular

[1]`http://www.novabase.pt/pt`

time series forecasting competition, Makridakis Competition (M-Competition) [57, 58];

- some approaches' computational complexity is lower;
- they are more robust than classical approaches;
- they enable learning of multiple time series, for the same problem.

Given these attributes, it makes sense for Machine Learning (ML) approaches to participate in reliable time series analysis. Going back to the topic where ML approaches are more robust, one has to highlight that there is no single method which performs universally better, for whichever time series and whichever features it embeds. In most cases, the data to be fed to a model has to undergo specific data preparation techniques in order to be correctly processed for the models it is applied to, and even then, there are a plethora of models that can be configured appropriately.

At the Centre for Cognitive Computing, approaches, pre-processing measures and evaluation techniques to integrate into an Auto-ML framework, based on the concepts of Case-based Reasoning (CBR) and Bayesian Optimization, were developed. The approaches to include in order to address both forecasting and anomaly detection in time series, the CBR paradigm and Bayesian Optimization are detailed in the state of the art.

1.3 Goals

The main objective in this project is to develop a framework that houses approaches to deal with time series analysis, in respect to Forecasting and Anomaly Detection. This framework autonomously selects the most appropriate model, using CBR, leaving as little data preparation, model selection and parametrization decisions to the user as possible, as this process is extensive. With this in mind, the framework performs these decisions based on the given time series' features and the search space specified for each approach, during hyperparameter tuning. The most relevant goals in the internship are:

- Support for data stemming from local files;
- Data processing, in order to conform to the models in use;
- Provide a framework, both for forecasting and anomaly detection, that manages approaches that execute those types of analysis, selects the most appropriate method to use, and parametrizes it, with quality maximization in mind.

Additionally, performance goals are also taken into account:

- The approach selection weights on a threshold used to relate quality to performance;
- The integrated approaches' results consider benchmarks to better their accuracy.

1.4 Outline

Chapter 2 - State of the Art: Describes a collection of the most relevant and recent concepts that helps understand how time series analysis has been analyzed throughout the years. The enumerated methods not only detail background knowledge necessary to better understand them, but are also provided with significant use cases which help reassure their strength in the literature. This organization is based on the analysis type that is most frequent in the literature: regarding Forecasting, both classical and machine learning models are provided; In Anomaly Detection, most of the presented approaches are ML approaches. After the described models for each analysis type, two key concepts used on the proposed approach are described: CBR and Bayesian Optimization. To conclude this chapter, a list and comparison of competitors who provide automation in time series analysis, for forecasting and anomaly detection, is presented;

Chapter 3 - Resources and Tools: This chapter highlights the resources and tools that are available and were used for the development of the proposed framework;

Chapter 4 - Requirements and Risk Analysis: Includes the motivation that drives the internship proposal as well as the methodology that will be used. Details on requirements specification are also present;

Chapter 5 - Approach: Reformulates the enumerated requirements as an architecture and provides technical details on the development of the proposed approach, organized in relation to the differences expressed between Univariate Time Series (UTS) and Multivariate Time Series (MTS) approaches;

Chapter 6 - Experimental Setup: Describes the pipelines of model selection and model hyperparameter tuning for each scenario previously described in the Approach Chapter. For each scenario, results analysis on both components is then provided;

Chapter 7 - Methodology and Planning: Presents an overview on the entire project in the context of time estimation for each of the tasks that compose this curricular unit. This chapter also details actual costs registered so far, during the course of the internship;

Chapter 8 - Conclusions: Finalizes this document with an overview on the research that led to the approach, a discussion over the results drawn from it and a reflection on future work.

Chapter 2

State of the Art

Time series are defined as "*a collection of observations made sequentially through time*"[23]. Examples occur in a variety of fields, from economics to engineering. Methods of analyzing time series constitute an important area of statistics.

As seen in [23], Chatfield differentiates time series categories into 6 different types:

- Economic and Financial Time Series;
- Physical Time Series;
- Marketing Time Series;
- Process Control Data;
- Binary Processes;
- Point Processes.

Categorization of time series becomes a necessity as there is no approach with the required generalization capabilities to address each and every type of time series. To better classify time series, one should consider the following features, as seen in [23]:

Seasonality: Whether the data presents a pattern that repeats itself over a time interval. For example, retail sales tend to peak during Christmas season and then decline after the holidays. Thus, time series of retail sales will present increasing sales from September to December and decreasing sales in the following months. Seasonality is quite common in economic time series, but not as much as in engineering and scientific data.

Trend: A general systematic linear or (most often) nonlinear component that changes over time but never repeats, or at least does not repeat for the recorded time range of the series.

Outliers: Observation points that are distant from other observations. Usually, the presence of outliers indicate measurement error or that the population has a heavy-tailed distribution. In the context of time series, for most cases, outliers represent anomalies in the captured data that closely relate to the nature of the time series. Assuming a time series that holds frequency data on an EEG scan, events such as an epileptic seizure set off a spike. If the series' density is high enough, a seizure may represent a fraction of the data. Thus, seizures are regarded as outliers in the distribution.

In [21], the authors reason there is a multitude of time series encountered in the fields of engineering, science, sociology and economics. The purpose of time series analysis is to draw inference from such structures.

After an appropriate collection of models is made, it is possible to estimate parameters, check for goodness of fit to the data, and use the fitted models to deepen our understanding of series generation. These models may be used to provide description of the data (through time plots), separate (filter) the noise from signals and test hypotheses such as global warming, using recorded temperature data.

There are several possible objectives in analyzing a time series [23]. These objectives may be classified as description, explanation, prediction and control.

Description: Usually, the first step in time series analysis is to plot the observations against time to obtain a time plot. The power of the time plot as a descriptive tool is present in holiday sales data where a regular seasonal effect, with sales 'high' in winter and 'low' in summer is frequent. Additionally, in some cases, annual sales increase (there is an upward trend). For data originated from EEG scans, time plots will often present epileptic seizures as spikes in the variable (frequency) value, suggesting that the data composing the time series is inconsistent, and these observations represent outliers.

Explanation: When observations are taken on two or more variables, it is possible to use the variation in one series to explain the variation in another series. This may lead to a deeper understanding of the series generation mechanism.

Prediction: Given an observed time series, one may want to forecast the future values of the series. This is an important task in sales forecasting, and in the analysis of economic and industrial time series.

Control: Time series are sometimes used to improve control over some physical or economic system. When a time series that measures the 'quality' of a manufacturing process is generated, the aim of the analysis is to keep the process operating at a highly efficient level. Control problems are closely related to prediction in many situations. If one can predict that a manufacturing process is going to move off target, then appropriate corrections can be applied.

In that sense, the following chapter sheds information on the available and state of the art approaches for time series analysis regarding forecasting and anomaly detection. Both a description on the available approaches and related work are provided.

2.1 Forecasting

Forecasting consists in predicting unseen values within an observed time series. For instance, if we have an observed series $x_1, x_2, ..., x_n$, the goal would be to estimate values , x_{n+h}, where h represents the *forecasting horizon.* This constitutes an important problem in areas such as economics, stock control, marketing and production planning.

While there is a wide variety of forecasting procedures, the generalization capability of the available methods has yet to reach an appropriate value [58]. Thus, there is a need to choose the most appropriate approach given a set of pre-conditions (such as those presented in the introduction of this chapter). It is also worth mentioning that forecasting is a form of extrapolation, therefore bearing all the risks of such an activity. The larger the considered *horizon*, the more prone the approach is to increase the associated error. Thus, the analyst should always be prepared to modify them as necessary in light of any incoming information.

As a side note, it is imperative to refer the Makridakis Competition (M-Competition) [55, 56, 57, 58], which is referenced throughout this section of the document. The M-Competition has become a benchmark in time series forecasting. The competitions are led by Spyros Makridakis and intend to evaluate and compare different approaches for forecasting time series.

The following subsections present state of the art approaches on forecasting. A brief description and application are provided for each of them.

2.1.1 Classical Models

The classical models correspond to the first, more statistical, approaches to be applied in forecasting. In this subsection, popular classical time series forecasting approaches are presented. Both a description and usage of the most prominent approaches (in this case, Autoregressive Integrated Moving Average (ARIMA) and Holt-Winters Exponential Smoothing (HW)) are highlighted. All models until the ARIMA assume that the time series are stationary [23].

Regression

Regression is a basic and commonly used type of predictive analysis.

There are a variety of techniques for modeling and analyzing multiple variables (from simple linear regression to discriminant analysis), where the focus is on the relationship between a dependent variable and one or more independent variables (or ‘predictors’). When a regression is used to give an expected value for a known value of a predictor, it is dubbed as a prediction. This prediction is obtained from a range of observed data points. A forecast is obtained from extrapolation of the observed range to predict an unseen observation. Regression analysis is widely used for both prediction and forecasting, where its use has substantial overlap with the field of Machine Learning. The simplest form of the regression equation, with one dependent variable and one independent variable, is defined as follows [50].

$$y = c + bx \tag{2.1}$$

where y is the estimated dependent variable score, c is a constant, b is the regression coefficient and x is the score on the independent variable. An graphical example of a regression is visible in Figure 2.1.

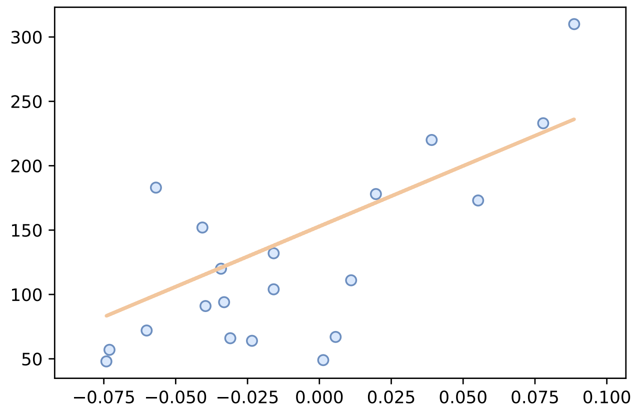


Figure 2.1: Regression applied on the Scikit-learn diabetes dataset.

Autoregressive Model

According to [23], the idea behind Autoregressive (AR) models is to obtain the present value of the series, X_t, by a function of p past values, X_{t-1}, X_{t-2}, ..., X_{t-p}.

An AR process of order p is written as

$$X_t = \alpha_1 X_{t-1} + \alpha_2 X_{t-2} + ... + \alpha_p X_{t-p} + Z_t \tag{2.2}$$

where Z_t is White Noise distribution, $\{Z_t\} \sim WN(0, \sigma^2)$, and Z_t is uncorrelated with X_s for each $s < t$.

White noise is also called a purely random process. A purely random process consists of a sequence of random variables, Z_t, which are mutually independent and identically distributed. Normally, it is further assumed that the random variables are normally distributed with mean zero and variance σ_Z^2.

This is rather like a multiple regression model, but X_t is regressed on past values of X_t rather than on separate predictor variables, which explains the prefix 'auto'.

Moving Average Model

In an Moving Average (MA) process, the current value is given by an expression of current and past values of the random disturbances that form a white noise series [23]. MA differs from AR because each random disturbance propagates to future values in the series. An MA process of order q can be written in the form

$$X_t = \beta_0 Z_t + \beta_1 Z_{t-1} + ... + \beta_q Z_{t-q} \tag{2.3}$$

where $\{Z_t\} \sim WN(0, \sigma^2)$ and β_i are constants. $\{Z\}$ is usually scaled so that $\beta_0 = 1$.

No restrictions on the β_i are required for a (finite-order) MA process to be stationary, but it is generally desirable to impose restrictions on the β_i to ensure that the process satisfies a condition called invertibility [23].

Autoregressive Moving Average

A useful class of models for time series is formed by combining AR and MA processes. An Autoregressive Moving Average (ARMA) model is suitable for univariate time series modeling. A mixed autoregressive/moving-average process containing p AR terms and q MA terms is said to be an ARMAprocess of order (p, q) and is given by

$$X_t = \alpha_1 X_{t-1} + ... + \alpha_p X_{t-p} + Z_t + \beta_1 Z_{t-1} + ... + \beta_q Z_{t-q} \tag{2.4}$$

In [2], ARMA model manipulation is made using the lag operator notation. The lag or backshift operator is defined as

$$\phi(B)X_t = \theta(B)Z_t \tag{2.4a}$$

where $\phi(B)$ and $\theta(B)$ are polynomials of order p and q, respectively, such that

$$\phi(B) = 1 - \alpha_1 B - ... - \alpha_p B^p$$

and

$$\theta(B) = 1 + \beta_1 B - ... - \beta_q B^q$$

The conditions on the model parameters to make the process stationary and invertible are the same as for AR or MA processes, namely, that the values of α_i, which make the process stationary, are such that the roots of

$$\phi(B) = 0$$

lie outside the unit circle, while the values of β_i, which make the process invertible, are such that the roots of

$$\theta(B) = 0$$

lie outside the unit circle.

Autoregressive Integrated Moving Average

In practice, many time series are non-stationary, and so the stationary models mentioned so far cannot be applied directly. In ARIMA modeling, the general approach is to differentiate an observed time series until it appears to come from a stationary process. Differenciating is widely used for econometric data. If X_t is replaced by $\nabla^d X_t$ in Equation 2.4, then we have a model capable of describing certain types of non-stationary series. Such a model is called an 'integrated' model because the stationary model that is fitted to the differenciated data has to be summed or 'integrated' to provide a model for the original non-stationary data, as seen in [23].

Writing

$$W_t^d = \nabla^d X_t = (1 - B)^d X_t$$

the general ARIMAprocess is of the form

$$W_t = \alpha_1 W_{t-1} + ... + \alpha_p W_{t-p} + Z_t + ... + \beta_q Z_{t-q} \tag{2.5}$$

By analogy with equation 2.4a, we may write the ARIMAprocess (2.5) as

$$\phi(B)W_t = \theta(B)Z_t \tag{2.5a}$$

or

$$\phi(B)(1 - B)^d X_t = \theta(B)Z_t \tag{2.5b}$$

Thus, we have an ARMA(p, q) process for W_t, while the model in equation 2.5b, describing the dth differences of X_t, contemplates an ARIMAprocess of order (p, d, q). The model for X_t is non-stationary, as the AR operator $\phi(B)(1 - B)^d$ has d roots on the unit circle. Usually, first differenciating is adequate to make a series stationary.

ARIMA was first published in [17] and quickly found its use for non-stationary data, like economic and stock price series. The ARIMAmodel and its different variations are based on the famous Box-Jenkins principle [8, 10, 38, 59, 69, 79] and so these are also broadly known as the Box-Jenkins models.

In [32], the airline data is used as the main example in a case study of forecasting results from a variety of neural network models against the Box-Jenkins seasonal ARIMAmodel referred to as the airline model.

In [10], findings show that modeling crude palm oil price through an ARIMAprocess is appropriate for short term predictions.

In [79], a comparison between an automatic Box-Jenkins modeling expert system, AUTOBOX, and a Neural Network was carried out, using a portion of time series stemming from the famous M-Competition [55]. Results showed that there was promise in using neural network models as these appeared to present similar error (Mean Absolute Percentage Error (MAPE)) in the generality of the used time series.

Several articles [8, 38, 69] also sought out to estimate machine learning methods' performance in relation to traditional methods, always including ARIMA in their experiments. Conclusions in this study wager traditional methods' appropriation in time series on annual data, while the neural networks in use are more suitable for monthly and quarterly data, as well as for unstable data (namely discontinuities).

Holt-Winters Exponential Smoothing

Exponential smoothing's application in forecasting was first introduced by Robert Brown in [33]. In 1957, Professor Charles C. Holt (1921-2010), MIT and University of Chicago graduate, was working at the Carnegie Mellon University on forecasting trends in production, inventories and labor force.

Holt published "*Forecasting trends and seasonals by exponentially weighted moving averages*" describing double exponential smoothing. Three years later, Peter R. Winters, his student, improved the algorithm by including seasonality and published "*Forecasting sales by exponentially weighted moving averages*", citing Holt's 1957 paper. Thus, this algorithm became known as triple exponential smoothing or HW.

The following paragraphs provide insight on the process from Single Exponential Smoothing to Triple Exponential Smoothing.

Single Exponential Smoothing Also known as simple exponential smoothing, it is used for short-range forecasting. This model assumes that there is no trend associated to the data. The formula is defined as follows, according to [1].

$$S_t = \alpha * y_t + (1-\alpha) * S_{t-1} \tag{2.6}$$

where S_t stands for the smoothed observation, y_t is the original observation and α represents the smoothing constant. In this model, the new predicted value is based on the old one in addition to an adjustment for the error associated with the last forecast.

Double Exponential Smoothing The Single Exponential Smoothing does not prove to be an appropriate method when the data follows a trend. This case can be corrected by introducing a new equation, with a second constant, γ. In Double Exponential Smoothing, both *level* and *trend* are updated for each period t.The level is an estimate of the data value at the end of each period, while the trend is an estimate of the average growth at the end of each period. The resulting equations are as follows, according to [1].

$$S_t = \alpha * y_t + (1-\alpha) * (S_{t-1} + b_{t-1}) \qquad 0<\alpha<1 \tag{2.7}$$

$$b_t = \gamma * (S_t - S_{t-1}) + (1-\gamma) * b_{t-1} \qquad 0<\gamma<1 \tag{2.8}$$

Note that the current value of the series is used to compute its smoothed value replacement in double exponential smoothing.

The initial value for S_t is generally y_1, while, for b_1, in [45] the following suggestions are presented:

- $b_1 = y_2 - y_1$
- $b_1 = [(y_2 - y_1) + (y_3 + y_2) + (y_4 - y_3)]/3$
- $b_1 = (y_n - y_1)/(n - 1)$

Triple Exponential Smoothing When the data shows both trend and seasonality, double smoothing is not enough. Thus, a third equation that takes care of seasonality is introduced. The resulting set of equations is named the "Holt-Winters" method, named after the inventors. The basic equations for this method are given by

$$S_t = \alpha \frac{y_t}{I_{t-L}} + (1-\alpha)(S_{t-1} + b_{t-1}) \tag{2.9}$$

$$b_t = \gamma(S_t - S_{t-1}) + (1-\gamma)b_{t-1} \tag{2.10}$$

$$I_t = \beta \frac{y_t}{S_t} + (1-\beta)I_{t-L} \tag{2.11}$$

$$F_{t+m} = (S_t + mb_t)I_{t-L+m} \tag{2.12}$$

where y is the original observation, S is the smoothed observation, b is the trend factor, I is the seasonal index, F is the forecast at m periods, t is an index denoting a time period and α, β and γ are constants that must be estimated with the goal of minimizing the Mean Squared Error (MSE).

To initialize Holt-Winters we need at least one complete season's data to determine initial estimates of the seasonal indices I_{t-L}. A complete season's data consists of L periods. To estimate the trend factor, is it advisable to use two complete seasons, that is, $2L$ periods.

Holt-Winters Exponential Smoothing is a prominent traditional method with application in time series forecasting. In [45], a detailed analysis on the weight of both multiplicative and additive models of HW when confronted with multiplicative and additive seasonality is made, obtaining an MAPE as low as 17.48%. Authors [8, 38] draw comparisons not only between machine learning methods and ARIMA, but also HW. In [48], a modified HW method where the forecasting horizon, m, is square rooted, leads to a more conservative trend extrapolation, and is pitted against Feedforward Neural Network (FFNN). Results showed that the best configured FFNNs obtained an Average Relative Mean Quadratic Error (ARMQE) comparable with that of a commonly configured HW model.

2.1.2 Machine Learning Models

While Machine Learning approaches have been theorized long ago, their application in the domain of forecasting is relatively recent. However, one can assume that considering the state of the art on time series analysis, Machine Learning is present and highlighted in the methods that follow in this subsection.

Multilayer Perceptron

The Multi Layer Perceptron (MLP) finds its base on the Perceptron algorithm. The **Perceptron** is a linear binary classifier for supervised learning. In that sense, classification is carried out through a linear prediction function which combines a set of weights and bias with a feature vector.

In [63], the authors define that "a perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise." Given inputs x_1 through x_n, the output $o(x_1, ..., x_n)$ computed by the perceptron is

$$o(x_1, ..., x_n) = \begin{cases} 1, & if\ w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n > 0 \\ -1, & otherwise \end{cases}$$

where each w_i is a real-valued *weight*, that determines the contribution of input x_i to the perceptron output. The quantity $(-w_0)$ is a threshold that the weighted combination of inputs $w_1x_1 + ... + w_nx_n$ must surpass in order for the perceptron to output 1. To simplify notation, an additional constant $x_0 = 1$ is introduced, such that the above inequation is rewritten in the form $\sum_{i=0}^{n} w_ix_i > 0$, or in vector form, $\vec{w} \cdot \vec{x} > 0$.

Single perceptrons can only express linear decision surfaces. However, multilayer networks learned by the backpropagation algorithm are capable of expressing a rich variety of non-linear decision surfaces. The MLP , at its most bare form, is a set of perceptrons with one additional layer of nodes (referred to as hidden layer). More complex variations include more than one hidden layer, as the MLP is a heavily parametrized model. The general representation for the MLP is show below:

$$\hat{y} = \upsilon_0 + \sum_{j=1}^{NH} \upsilon_j g(w_j^T x') \tag{2.13}$$

where x' is the input vector x, augmented with 1, i.e., $x' = (1, x^T)^T$, w_j is the weight vector for jth node, $\upsilon_0, \upsilon_1, ..., \upsilon_{NH}$ are the weights for the output node, and $\hat{y}$ is the network output. Function g represents the hidden node output, and is given in terms of a squashing function.

The MLP uses a supervised learning technique called backpropagation for training. The backpropagation algorithm learns the weights, given a network with a fixed set of units and interconnections. In order to minimize the squared error between the network output and target values, it employs gradient descent.

The error in output node j, for the n^{th} training example, is given by

$$e_j(n) = t_j(n) - o_j(n)$$

where t, o are the target and output values. The node weights are adjusted based on corrections that minimize the error in the entire output:

$$\varepsilon(n) = \frac{1}{2} \sum_j e_j^2(n)$$

Using gradient descent, the change in the weights is given by

$$\Delta w_{ji}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial \upsilon_j(n)} y_i(n)$$

where y_i is the output of the previous neuron and η is the learning rate.

The MLP has a strong presence in the state of the art of time series forecasting. [69] presents a case study discussing the comparison between supervised artificial neural network and ARIMA models. As part of the analysis, one of the first problems to occur was the search of appropriate input lag terms for training the network. This led the experiment to account for performance of different NN models with varying order of the network, number of parameters and activation function in use. Results showed that the the most appropriate model in the fitted Neural Network (NN) models showed a lower SSE than the ARIMAX fitted model.

In [6], a large scale comparison study for major machine learning models, including MLP , for time series forecasting took place, using time series provided from the M3 Competition. While different preprocessing techniques were approached, both MLP and Bayesian Neural Network (BNN) stood out as presenting the best overall results. In this study, the performance metric in use was the sMAPE-TOT. First, each model ran 10-fold validation, so that an overall Symmetric Mean Absolute Percentage Error (sMAPE) for each model could be obtained. The sMAPE-TOT accounts for the total sMAPE on all 1045 considered time series. The average sMAPE is computed from the validations. Result analysis led to believe that for each time series category, MLP consistently presented the lowest (best) sMAPE-TOT. As an addition, a time complexity study was provided, showing that while MLP presented the best sMAPE-TOT, it was not necessarily the fastest.

In [46], the correlation between crude palm oil price (CPO), selected vegetable prices, crude oil and the monthly exchange rate is explored. Preliminary analysis showed a positive, high correlation between CPO price and soy bean oil price, and CPO price and crude oil price. The undergoing experiments used MLP , Support Vector Regression with Sequential Minimal Optimization (SMO) and HW. Error metrics in use include Root Mean Square Error (RMSE), Mean Absolute Error (MAE), MAPE (and Direction Accuracy (DA) as a means to understand whether the forecast value tends in the same direction as the series). In this study, SMO was consistently more accurate than MLP , but the worst model in use was HW.

As expected, MLP made its way onto the M3-Competition. In [59], Makridakis et al. assess machine learning models' disadvantages when compared to classical approaches, both in forecasting accuracy and computational complexity. In that sense, this paper proposes possible ways forward as well as an explanation for ML approaches' sub-par performance when compared to classical approaches. One of the major doubts is present in the error associated with MLP applied at the most appropriate preprocessing, when compared with that of ARIMA and the seasonally adjusted random walk model, which appear to perform marginally better. One way to improve forecasting on current ML approaches is to have the data deseasonalized. Another, is to have Machine Learning (ML) methods access information about surpassing the available data for training, with the objective being to minimize future errors rather than fitting the model to available data. Another important concern is the extent of randomness associated with the series and the ability of the ML models to differentiate noise from the data, all while avoiding over-fitting. This task can prove difficult as, unlike classical models, where noise is easily parametrized by information criteria (such as Akaike information criterion (AIC)), ML methods are non-linear and training is performed dynamically.

Bayesian Neural Network

Bayesian Inference is a statistical inference method in which Bayes' theorem is used to update probabilities in a hypothesis as more information is committed. The prior, $p(\theta)$, is the *prior* probability of a parameter θ before having seen the data. The likelihood, $p(D|\theta)$, is the probability of the data D given θ. Using Bayes' rule, we can determine the *posterior* probability of θ given the data, D, as

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \tag{2.14}$$

In general, this provides an entire distribution for values of θ rather than a single most likely value.

A BNN is a neural network based on Bayesian inference, thus related to the statistical concept of Bayesian parameter estimation as well as regularisation [15]. BNNs closely resemble MLP, but weights are observed as random variables which obey some prior distribution. According to [6], this distribution is designed to favor low complexity models. Following data observation, the posterior distribution of the weights is evaluated and the network prediction can be assessed. Predictions reflect both smoothness imposed through the prior and fitness accuracy imposed by the observed data.

As seen for MLP, the comparison study in [6] included BNNs. Both MLP and BNN stood out as presenting the best overall results, with the BNN fitted model only marginally worse than MLP. The provided time complexity study showed that BNN had a slower execution than MLP, suggesting that this approach, while interesting considering sMAPE values, might not be the most appropriate, at least for the considered time series.

The M3 Competition dataset once again provides insight into ML forecasting techniques, namely BNN [59]. Considering only ML approaches, BNN manages to obtain the lowest overall sMAPE, followed closely by MLP .

Radial Basis Function Neural Network

In the Perceptron and MLP, separability is linear, as they are composed of input and output layers (although MLP attains non-linearity because of hidden layers). Thus, to deal with non-linear separation, at least one hidden layer is necessary.

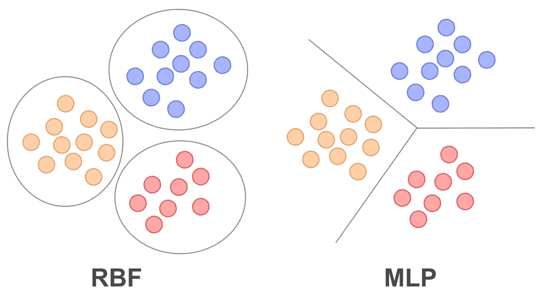


Figure 2.2: Distinction between MLP and Radial Basis Function Neural Network (RBFNN).

The RBFNN appears similar in structure to the MLP, but in its architecture only one hidden layer is allowed, as proposed in [22].

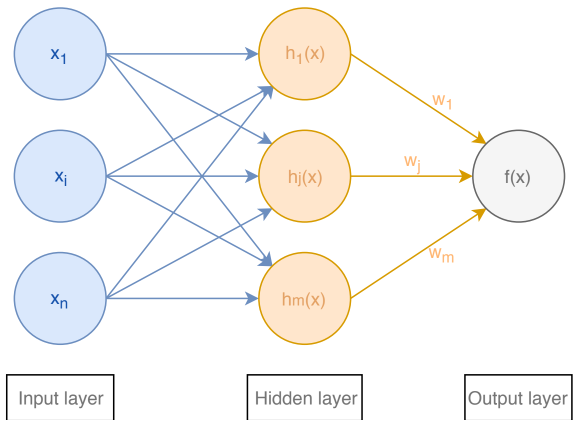


Figure 2.3: Traditional RBFNN.

Thus, as seen in figure 2.3, an RBFNN has exactly one input, one hidden and one output layer. Here, the hidden layer is typically referred to as feature vector and each of its neurons houses a basis function. The output of the RBFNN is a linear combination of the hidden nodes, defined as

$$y = \sum_{j=1}^{m} w_j h_j(x) \tag{2.15}$$

where w_j denotes the weight on hidden node j and $h_j(x)$ the basis function.

Any set of functions can be used as the basis set, although a well behaved set (differentiable) is preferred. Here, a special family of functions is highlighted: the radial functions. Radial functions' characteristic feature is that their response decreases (or increases) monotonically with distance from a center point. A frequently used radial function is the Gaussian which, in the case of a scalar input, is defined as

$$h(x) = exp\left(-\frac{(x-c)^2}{r^2}\right) \tag{2.16}$$

with parameters c being the center and r being the radius.

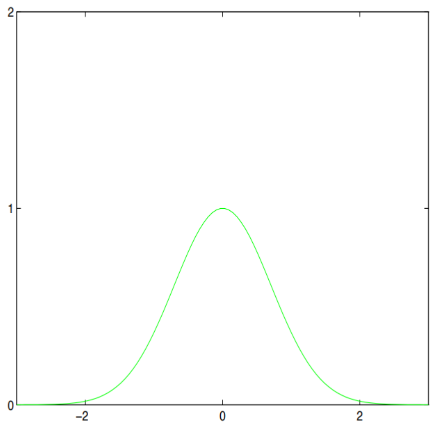


Figure 2.4: Gaussian RBF with center 0 and radius 1.

RBFNNs were first formulated in [22], as a mean to turn interpolation between known data points explicit. RBFNN learning is equivalent to solving a set of linear equations, and, while these networks represent non-linear relationships, they have a "*guaranteed learning rule*".

In [27], a demonstration on the use of neural networks in solar radiation modeling is presented. The NNs in use are the MLP and RBFNN. The RBFNN architecture in use comprises 5 input nodes (month of the year, latitude, longitude, altitude and sunshine ratio), hidden nodes use gaussian function and the output node's activation function is linear. Performance evaluation through the use of RMSE showed that performance of the different approaches are similar, but the RBFNNs are recommended as they "*do not need as much computing power as the MLP networks*".

As was the case for the previously highlighted ML approaches, in [6] the RBFNNs are also subject to comparison using the M3-Competition monthly series data. Although MLP and Gaussian Process (GP) present the best overall performance (sMAPE-TOT), it becomes clear in the analysis of computational complexity that RBFNNs present a clear advantage over MLP , while GP continue to execute in less time than the former two. It is important to emphasize the RBFNN's poor overall performance on all the used time series, for the MOV-AVG preprocessing method, achieving the worst sMAPE of all considered ML approaches. The study on the relation between computational complexity and performance metric could shed some information on which model to adopt, given cases where this multi-objective scenario needs to be handled.

While the previous use case denoted this method's poor performance, things seem brighter for the study conducted on the M3-Competition dataset, as seen in [59], where the authors compare their results to the ones obtained in [6]. While the sMAPE has improved greatly, computational complexity seems to have degraded, now being worse than both MLP and GP. These results account only for the one step ahead forecasting horizon, but considering the most appropriate preprocessing method per considered approach.

k Nearest Neighbor Regression

k-Nearest Neighbours (k-NN) is a non-parametric regression method that bases its forecasts on a similarity measure (distance function) from the new data points to the training set and outputs a prediction value based on the average of the k nearest neighbors [63]. This approach is considered a lazy algorithm because it does not learn a discriminative function, but instead stores the entire training dataset. In a logistic regression algorithm, training time entails model weight learning, while the same does not happen for k-NN. Thus, training time in k-NN is null. However, the prediction process is costly: every time we want to make a prediction, the algorithm computes the similarity for the k nearest neighbors, with the prediction being the result of an average on the previous k closest points. The most common distance functions are shown below.

$$D_{Euclidean} = \sqrt{\sum_{i=1}^{k}(x_i - y_i)^2} \tag{2.17a}$$

$$D_{Manhattan} = \sum_{i=1}^{k}|x_i - y_i| \tag{2.17b}$$

$$D_{Minkowski} = \left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{\frac{1}{q}} \tag{2.17c}$$

Handling regression problems, it seems logical for k-NN to be targeted for analysis on time series forecasting. In [6], k-NN is compared with several other ML approaches. Results show that for the overall performance on all the used time series, this approach presents one of the worst results of the comparison, while the MLP stands out. However, as expected, the computational complexity of this algorithm is very reduced. Results on this variable show that k-NN has the lowest computation time of all methods.

In the M-3 Competition, results analysis also befalls on k-NN [59]. Yet again, the forecasting performance on this method falls short of the competition, scoring the fourth to worst overall result, only slightly better than Long Short Term Memory (LSTM), but still worse than the Ahmed's study. Regarding computational complexity, it is shown again that k-NN executes quickly, even though its accuracy (sMAPE) is slightly degraded.

In [52], the goal is to reduce the cost related to management of the monitoring systems on solar radiation. To achieve this, experiments entailing plausability on the short term prediction of the solar radiation based on data collected in the near past on the same site and at weather stations located 10 kilometers away are conducted. The prediction techniques in use are Support Vector Regression (SVR), Extreme Learning Machine (EML) and Autoregressive models. these approaches are then compared with persistence and k-NN predictors. When comparing the presented approaches with k-NN, results show that k-NN is able to achieve performance comparable with more sophisticated models.

Classification and Regression Trees

Decision Trees are an important type of algorithm for machine learning prediction modeling. The classical decision tree algorithms have been around for decades and modern variations are among the most powerful techniques available. Classification and Regression Trees (CART) is a classification or regression model based on a hierarchical tree-like partition of the input space [19]. The input space is divided into local regions identified in a sequence of recursive splits. The tree structure consists of internal nodes and terminal leaves. Given a data example, the tree is traversed, through binary decisions, from the root node to a final leaf node, where a prediction is made.

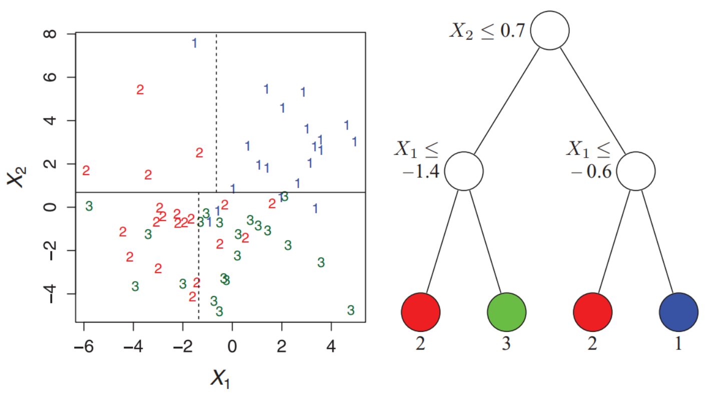


Figure 2.5: Input space partitions (on the left) in relation to the tree structure (on the right). Based on [54].

The CART was originally proposed in [19]. Its usage in time series forecasting was frequently highlighted in papers that were previously referenced in this document, in [6] and [59]. In the former, CART sMAPE positioned the algorithm at the second to worst result, while for the latter, it presents average results, considering ML methods.

Support Vector Regression

Support Vector Machine (SVM) (proposed in [16]) integrate a specific class of algorithms, characterized by their usage of kernels, absence of local minima, sparseness of the solution and capacity control obtained by parametrization of the margin and the number of support vectors. SVMs are frequently used in classification problems where different classes are not linearly separable at their original dimensionality. Thus, the use of kernels projects data features into a higher dimension where linear separation hyperplanes can be achieved. The capacity of the system is controlled by parameters that do not depend on the dimensionality of feature space.

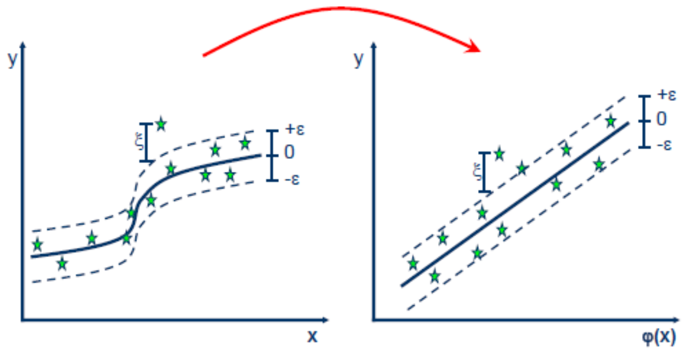


Figure 2.6: SVM kernel projects the data into a higher dimensional feature space to allow linear separation. Based on [78].

SVMs can be applied not only to classification problems, but also regression. SVR is the regression process performed by a SVM which tries to identify the hyperplane that maximizes the margin between two classes while minimizing the total error under tolerance. The error function is given by

$$J = \frac{1}{2}\|w\|^2 + C\sum_{m=1}^{M} |y_m - f(x_m)|_e \tag{2.18}$$

where the first term penalizes model complexity and the second term describe the ϵ-insensitive loss function, defined as

$$|y_m - f(x_m)|_e = max\{0, |y_m - f(x_m)| - \epsilon\} \tag{2.19}$$

Errors below ϵ are not penalized, allowing the parameters to adjust, in order to reduce model complexity.

SVRs were first documented in [28] and have had a frequent presence in the domain of time series forecasting. Recurrent references, [6] and [59], included the SVR in their comparison study of classical and machine learning approaches. Results on the former portray a reasonable overall performance, for the LAGGED-VAL pre-processing, while the remaining choices deteriorate SVR and most of the methods' sMAPE. For computational complexity, the SVR also stands reasonable, at fifth for the longer computation time. Regarding the latter, the used configuration showed notable improvements. The kernel in use is the radial basic one, ϵ is set to the noise level of the training sample while C is fixed to the maximum of the target output values. Next, both the γ parameter and number of inputs N are optimized through 10-fold validation. Results show promise of this approach, as it stands right behind MLP and BNN which consistently obtain good results for pure ML methods.

In [52], SVRs were one of the implemented models to pitch against the persistence and k-NN predictors. Parameter optimization (input dimensionality, ϵ and regularization trade-off) was done ad-hoc and the effectiveness was assessed through cross validation. SVR appropriation was tested in the second experiment (remote measurements) through the prediction error defined within this paper. The model with the lowest achieved error was the best SVR model (error of 40.5, when compared to the best k-NN which scored 41.4, followed by the best EML model, with error 42.7).

In [86], an overview on machine learning methods used for prediction is given. Methodologies such as regression trees, random forest and gradient boosting are highlighted to show that more methods, other than the popular neural networks and SVM/SVR, are being used in the context of forecasting, in the environment of solar radiation. In the conclusions, the authors assess SVM/SVR frequency, noting that in the following years they expected this method's use to rise, given that "*results (...) are very promising and some interesting studies will certainly be produced the next few years*".

In [46], the aim was to study variable correlation (crude palm oil price, selected vegetable oil prices and others), followed by price forecasting. Methods in use for forecasting included MLP , HW and SVR. In this particular case, the authors used a variant proposed in [81] referred to as SVR with

SMO, with the consideration that "*SMO has the good ability to model regression, prediction with non-linear data*". Results show that, for all considered performance metrics (MAE, DA, MAPE, RMSE and MSE), SMO stood out with considerably lower error.

Generalized Regression Neural Network

The Generalized Regression Neural Network (GRNN), also called Nadaraya-Watson estimator or kernel regression estimator was first proposed by Nadaraya and Watson in [68, 88]. The machine learning implementation (usually referred to as GRNN) was proposed by Donald Specht in [83]. The GRNN is a non-parametric model where predictions are computed by averaging the target outputs of the training dataset points according to their distance from the observation.

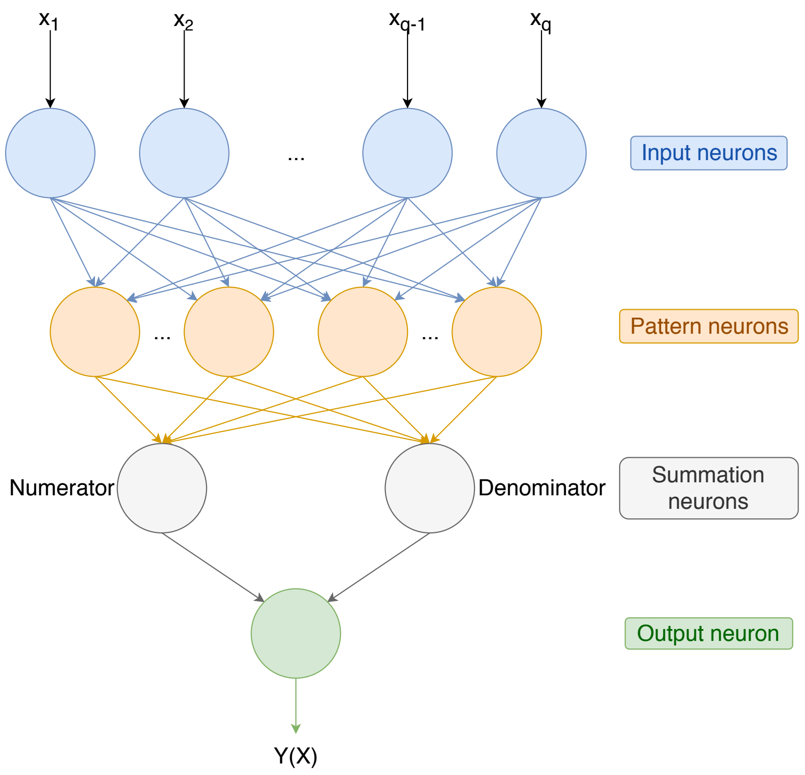


Figure 2.7: GRNN built for use as a parallel Neural Network. Based on [13].

As shown in [6], the estimation is the weighted sum of the observed responses, given by

$$\tilde{y} = \sum_{m=1}^{M} w_m y_m, \tag{2.20}$$

where the weights, w_m, are given by

$$w_m = \frac{\mathfrak{K}\left(\frac{\|x-x_m\|}{h}\right)}{\sum_{m'=1}^{M} \mathfrak{K}\left(\frac{\|x-x_{m'}\|}{h}\right)}, \tag{2.21}$$

where y_m is the target output for training data point x_m, and $\mathfrak{K}$ is the kernel function. The bandwidth, h, is an important parameter as it determines the smoothness of fit.

The GRNN is similar in structure to the RBFNN , having the following properties in common:

- One-pass learning, such that no backpropagation occurs;
- High accuracy in estimations, as gaussian functions are used;

- Robust to noise in the data.

However, as seen in the following use cases, the GRNN also has some disadvantages:

- It can increase in size, as the number of pattern neurons should be the same as the number of input neurons. This leads to a high computational complexity;
- There is no optimal method to improve this structure.

GRNNs have been explored in the literature. In [6], the gaussian kernel function is used. In order to assess the h parameter fit, 10-fold validation is performed. Results highlight the GRNN (and the CART) as the method with the best rank out of all the tested methods, for the DIFF pre-processing method. Generally, the GRNN presents satisfactory results, placing this approach in the middle of the ranking of all considered methods. Regarding computational complexity, the GRNN holds the third place for fastest executing algorithm.

The same methodology for parameter selection is adopted in [59], for both the h parameter and number of inputs N. Implementation details state this method was designed using the `R` statistical package. Results for the one-step-ahead forecasts with the most appropriate pre-processing technique per method show improvements over Ahmed et al. variant, decreasing the sMAPE by almost 1%, positioning this method behind Recurrent Neural Networks (RNNs) and in front of RBFNNs. Overall, the performance of this method shows promise, especially considering its low computational complexity.

Gaussian Process

In a simple linear regression, we have a dependent variable y. It is assumed that this variable can be modeled as a function of an independent variable, x through the expression

$$y = f(x) + \epsilon, \tag{2.22}$$

where ϵ is the error associated with the prediction. It is further assumed that function f defines a linear relationship, such that the objective is to find the slope and intercept of the line:

$$y = mx + b + \epsilon \tag{2.23}$$

In [11], it is highlighted that Bayesian linear regression "*provides a probabilistic approach to this by finding a distribution over the parameters*".

However, the GP approach is non-parametric, in the sense that it finds a distribution over the possible functions $f(x)$ that are consistent with the observed data. As with Bayesian methods, it starts with a prior distribution and updates the data as points are observed, outputting posterior distributions over functions. A GP assumes that $p(f(x_1), ..., f(x_N))$ is jointly Gaussian, with some mean $\mu(x)$ and covariance $\sum(x)$, given by $\sum_{ij} = k(x_i, x_j)$ where k is a positive definite kernel function. If x_i and x_j are deemed by the kernel to be similar, the output of the function is expected to be similar too. [67].

GPs have made recent appearance in the forecasting community, namely in popular studies [6] and [59]. In the former, parameter optimization follows the model selection algorithm proposed in [76], as it maximizes the marginal likelihood function. The authors state that such a criterion function does not favor complex models, thus overfitting is unlikely. Results show that while the LAGGED-VAL pre-processing presents the best results overall, GP sMAPE-TOT performed considerably well, independently of the pre-processing technique, always ranking in the top three lowest sMAPE methods. Data category results favor this model in regards to demographic time series. In regards to computation complexity, GP performs similarly to RBFNN , while BNN is severely slower in execution and k-NN is faster. Makridakis' study reveals a slight improvement in sMAPE, surpassing RNN , GRNN and RBF. This could be due to parameter (number of input variables, kernel initial noise variance and tolerance of termination) optimization, which undergoes 10-fold validation. The kernel in use is the radial basis.

Long Short Term Memory

To better understand the Long Short Term Memory Neural Network, one must first consider the Recurrent Neural Network [29]. The RNN, also known as Elman Network, is similar to MLP in structure, but includes a feedback loop, such that the output from step $n-1$ is fed back to the net to affect the output from step n and so forth, for each subsequent step. For example, if a RNN is exposed to a word letter by letter and a suggestion for the following letter is requested, the first letter of the word can help determine what the RNN thinks the second letter can be.

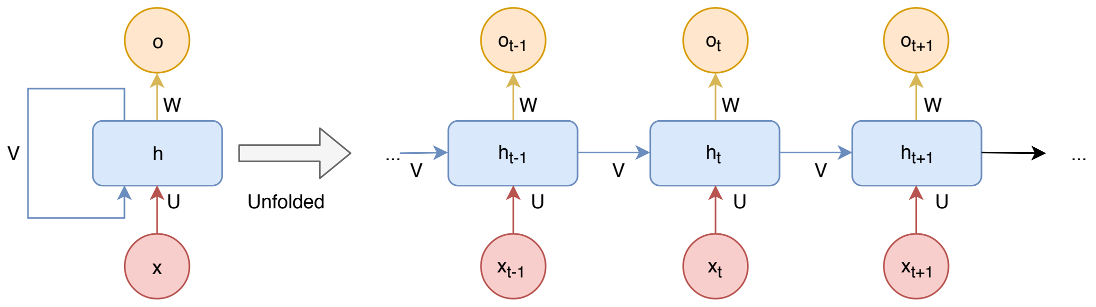


Figure 2.8: Unfolded RNNdiagram. Based on [70].

RNNs are a powerful set of NN algorithms especially useful for processing sequential data such as sound, time series or written natural language. RNNs and FFNNs both "remember" something about the data they are exposed to. After training, FFNNs produce static models of the training data that can classify new examples with acceptable accuracy. In RNNs, the models exhibit dynamic temporal behavior, so the classification always depends on the context of the observations they are exposed to. This is possible because of the RNNs hidden states that determine the previous classification in a series. In each subsequent step, that hidden state is combined with the following step's input data to produce a new hidden state and a new classification.

Just as neural networks as a whole take inspiration in the human brain, RNNs mostly weight on the human memory aspect. Human memories are context-aware, recycling previous awareness cases to properly interpret new situations. For example, assuming two people in the following circumstances: one is expecting a delivery from Amazon; the other has a close relative in the hospital. Both receive a phone call, and, while for the person expecting the delivery the thought is that the courier could not deliver the order, the person with the relative at the hospital might react worse. Interpretation for the phone call is different for each of them, because they retain a hidden state affected by their short-term memories and preceding sensations.

Different short-term memories should be recalled at different times and situations, in order to assign the correct meaning to the current input. Some of these memories are more recent than others. With this in mind, the RNN that binds memories and time input is called a LSTM.

The LSTM is similar to the RNN described above, only it avoids the long-term dependency problem of the former [39]. The evident advantage in LSTMs is their ability to keep information over long periods of time due to the structure of the units that compose them. These units consist of several gates that manage the weight attributed to the information in the current state.

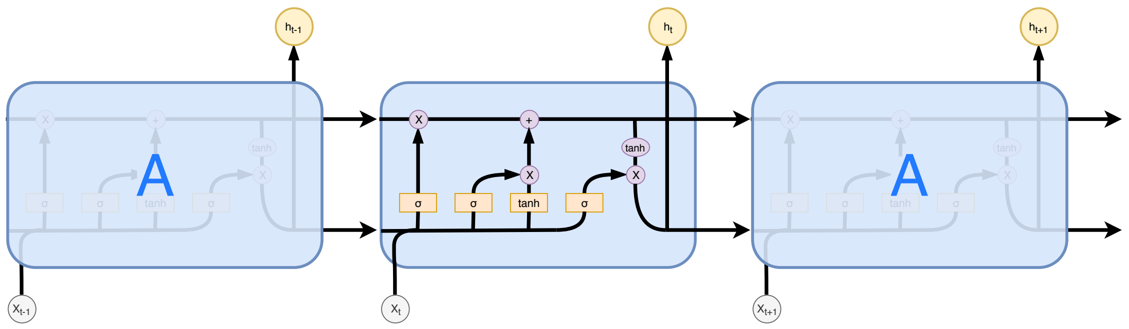


Figure 2.9: Module present in an LSTM. Based on [70].

As seen in figure 2.9, what differentiaties LSTMunits from RNNis the number of gates per unit.

In LSTMs, the cell state is the straight horizontal line running through the top of the diagram. The cell state is the core of the LSTMallowing for information to flow along the network with minimal changes. Using the results from the gates, it is possible to modify the information.

LSTM gates are structures that optionally let information through to the cell state. They incorporate a sigmoid neural net layer (with outputs between 0 and 1) and a point-wise multiplication operation.

The first gate layer manages what information is kept or thrown away and is dubbed *forget gate layer*.

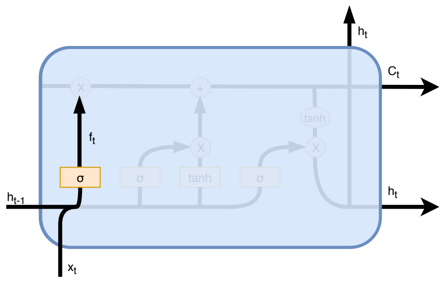


Figure 2.10: Module present in an LSTM- forget gate layer highlighted. Based on [70].

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.24}$$

It looks at the new input, x_t, and the previous module's output, h_t, and outputs a number between 0 and 1, for each number in cell state C_{t-1}.

The second gate layer decides the weight of new information on the cell state. This gate is called *input gate layer*.

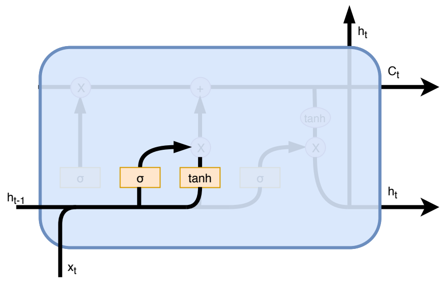


Figure 2.11: Module present in an LSTM- input gate layer highlighted. Based on [70].

First, a sigmoid layer decides which values to update.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.25}$$

Next, a *tanh* layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state.

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{2.26}$$

Finally, these vectors are combined to create an update to the state.

In the previous layers we decided what old information to keep and what new information to add. At this point, from the old state, C_{t-1}, the new state, C_t, is created.

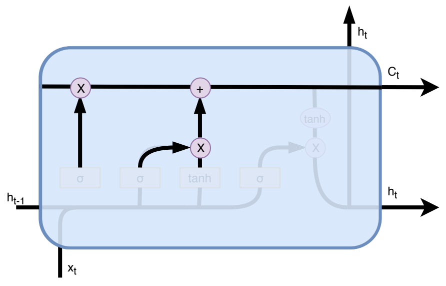


Figure 2.12: Module present in an LSTM- new state creation highlighted. Based on [70].

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t \tag{2.27}$$

The output from the module is now dependent on a filter from the new state. First, a sigmoid layer decides what parts of the cell state are output. Next, the cell state passes *tanh* to normalize the values between -1 and 1. Finally, these values are multiplied by the output of the sigmoid gate, so that only the wanted parts are output.

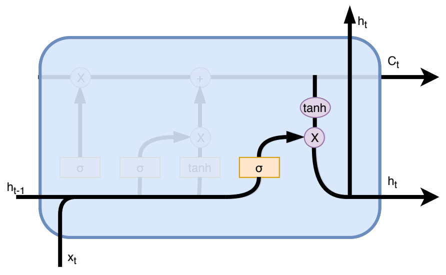


Figure 2.13: Module present in an LSTM- output gate layer highlighted. Based on [70].

$$\begin{aligned} o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= o_t \times tanh(C_t) \end{aligned} \tag{2.28}$$

The LSTM finds its origins in the late 90s, proposed by [39]. In [59], both RNNs and LSTMs are explored. The RNNin use has two layers: a hidden one containing recurrent nodes; an output one containing one or more linear nodes. Parametrization of this structure did not benefit from k-fold validation, as the computational requirements were too high.

As seen in the RNN modeling, the LSTM structure contains both a hidden and an output layer, without benefiting from validation as the computational complexity remains, as expected. For one-step-ahead forecasts, the RNN manages to perform reasonably, while the LSTM presents the third to worst ML method performance, considering the sMAPE metric. Regarding computational complexity, while MLP presents one of the best perfomances for ML methods, both RNN and LSTM are substantially faster in their result output. Another interesting result is the case of LSTMs, when compared to simpler NNs like RNN and MLP, reporting better model fitting (MSE normalized by the mean value of the time series being examined) but worse forecasting accuracy.

At Uber, the resurgence of LSTMs motivated [51] to propose a new LSTM-based architecture that outperforms the current state of the art event forecasting methods on Uber data and has an acceptable generalization capability to the public M3-Competition dataset. First, a strategy for uncertainty computation(both model and forecasting uncertainty) is presented, followed by a scalable neural network architecture for time series forecasting. From experience, the authors conclude that neural network models are most appropriate when the data size, data length and correlation among time series is high.

Not long after, one of the previous work's main contributors, Smyl, proposed a solution in the M4-Competition which ended up as the top resulting implementation [58]. The hybrid combination consists of an exponentially smoothed RNN. This method was particularly interesting because it incorporated information from both individual time series and the entire dataset, exploiting data in a hierarchical way.

Hidden Markov Model

A Markov Model, or Markov Chain, assumes that, from a sequence of observations, $X_1, ..., X_T$, of length T, X_t contains all the relevant information for predicting the future. Assuming discrete time steps, it is possible to write the joint distribution as follows [67].

$$p(X_{1:T}) = p(X_1)p(X_2|X_1)p(X_3|X_2)... = p(X_1)\prod_{t=2}^{T} p(X_t|X_{t-1}) \tag{2.29}$$

If we assume the transition function $p(X_t|X_{t-1})$ is independent of time, then the model is time-invariant, or stationary. This assumption allows modeling a number of variables using a fixed number of parameters. These models are called stochastic processes. Assuming that the observed variables are discrete, $X_t \in 1, ..., K$, this is called a finite-state Markov chain.

When X_t is discrete, the conditional distribution $p(X_t|X_{t-1})$ can be written as a $K \times K$ matrix, known as the transition matrix, A, where $A_{ij} = p(X_t = j|X_{t-1} = i)$ is the probability of transitioning from state i to state j. Each row of this matrix sums up to 1, $\sum_j A_{ij} = 1$, thus this is a stochastic matrix.

A finite-state Markov chain is usually visualized through a directed graph, where the nodes represent states and the arrows represent transitions. This is known as a state transition diagram. From here, it is also possible to describe the node connections through a binary matrix. Take the following example, in figure 2.14.

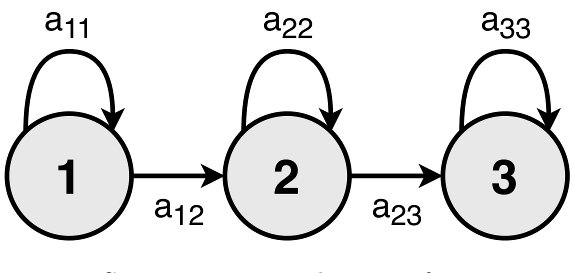


Figure 2.14: State transition diagram for a 3-state chain.

Then the transition matrix is as follows.

$$A = \begin{bmatrix} A_{11} & A_{12} & 0 \\ 0 & A_{22} & A_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

A **Hidden Markov Model (HMM)** is a discrete-state Markov chain, with hidden states, $x_t \in \{1, ..., T\}$, as well as an observation model, $p(x_t|y_t)$ [67]. The joint distribution is of the following form.

$$p(y_{1:T}, x_{1:T}) = p(y_{1:T})p(x_{1:T}|y_{1:T}) = \left[p(y_1) \prod_{t=2}^{T} p(y_t|y_{t-1}) \right] \left[\prod_{t=1}^{T} p(x_t|y_t) \right] \tag{2.30}$$

In addition to the state transition matrix, the HMM is also defined through an observation probability matrix, B, which details the probability of a hidden state, given the observed state. If we assume the following notation [75]:

T	= length of the observation sequence
N	= number of states in the model
M	= number of observation symbols
Q	$= \{q_o, q_1, ..., q_{N-1}\}$ = distinct states of the Markov process
V	$= \{0, 1, ...M-1\}$ = set of possible observations
A	= state transition probabilities
B	= observation probability matrix
π	= initial state distribution
$\mathcal{O}$	$= (\mathcal{O}_0, \mathcal{O}_1, ..., \mathcal{O}_{T-1})$ = observation sequence

We can illustrate a generic HMM in figure 2.15.

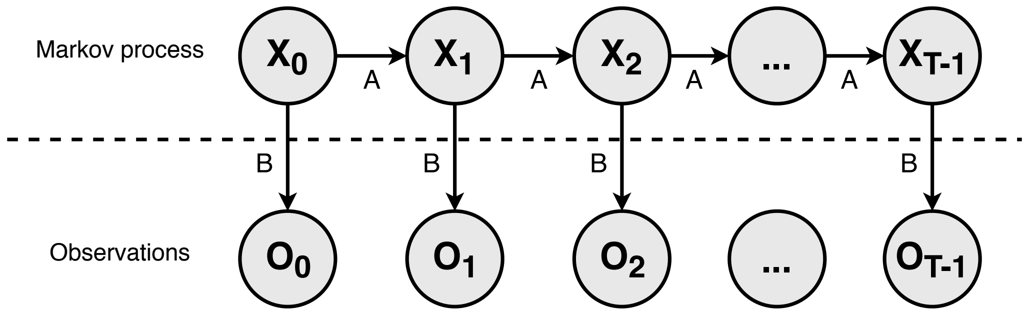


Figure 2.15: A Hidden Markov Model.

HMMs have also been considered in the literature, especially in cases where the data is stochastic.

In [91], a HMM with a non-Guassian distribution is proposed for application in ozone zone identification. In a HMM, the conditional distribution of the observed variable, Y_t, at the hidden state X_t is referred to as an emission distribution. Generally, this distribution is assumed to have a Gaussian distribution. However, the authors propose an estimation method for the shape and scale parameters of the Gamma distribution. The considered data stems from ozone concentration measurements, from monitoring stations at Livermore, near San Francisco, and the Houston Deer Park. Results were collected for two scenarios. The first considers ozone concentrations below and above 75 ppb. In this case, both a HMM-Gaussian and HMM-Gamma were applied, to each monitored area. Here, both models presents a perfect True Positive Rate (TPR), while the HMM-Gamma reduces the False Alarm Rate (FAR), in relation to the HMM-Gaussian by 77.42%, in Livermore. In Houston, the FAR decreases from 11.48% to 7.79%. The second scenario considers 3 ozone levels: from 0 to 57 ppb; from 58 to 75 ppb; 76 to 120 ppb. Then, comparisons between HMMs for two and three levels are drawn. For both level considerations, the HMM-Gamma presented better results, although the difference between HMM-Gaussian and HMM-Gamma for three levels is negligible. HMMs for three levels presented better results. Thus, the authors conclude that the number of levels impacts prediction performance more than the choice of emission distributions. In [74], a probabilistic approach using HMM is proposed, in order to provide short term prediction of traffic conditions on freeways, during peak periods. The authors' motivation for this approach is that change in traffic conditions over a short time period is stochastic. Thus, the randomness in freeway traffic conditions due to variation (in travel demand, weather, frequency of incidents and others) can be accounted for using a traffic state transition probability matrix. The traffic state is hidden, since one traffic parameter value can not fully reflect the prevailing traffic conditions. The data used in this study stems from a trace collected over a 61km corridor of Interstate 4 in Orlando, Florida. This corridor is known for experiencing heavy congestion during morning and evening peak periods. In order to assess model quality, the authors resort to a new method. Using the centroid to represent each traffic state, the method computes the ratio between the euclidean distance of the predicted state to the the observed state, as well as the longest distance between the observed state and the farthest state in the state space domain. The overall prediction error is measured by the average prediction errors for all applications of the prediction model. The experiment also included a sensitivity study of the HMMs to test if the models were site or time specific. As a term of comparison to the implemented HMMs, two naïve traffic prediction methods were used. The first uses the current traffic state for prediction, while the second uses traffic speed to describe traffic condition. Results show that errors from the HMMs are consistently lower than those from first method, regardless of the time of peak period. When the data becomes noisy, the HMMs manage to present errors significantly lower than the first method. Error comparison between the HMMs and the second method present similar trends. However, when the HMMs errors increase, their growth rate is higher for the second method. Thus, the HMMs show robustness in handling noisy data.

2.1.3 Remarks

Regarding ML models, there is a clear advantage when comparing the presented models. In [59] this is especially evident, as all shown and discussed models are built and tested on the same data set. When considering forecasting through machine learning, one should look first for MLP, BNN and SVR. The use cases for LSTM architectures also present interesting results and should be worth investigating in the context of the project associated with this document.

2.2 Anomaly Detection

Anomaly Detection (also often called outlier or novelty detection) is "*an important data analysis task that detects anomalous or abnormal data from a given dataset*" [4]. Outliers are important because they indicate rare events that usually prompt critical actions to address in a variety of domains. Recently, the number of computer systems (desktop devices, such as smart TVs and computers, and mobile devices, much like laptops and smartphones) has come to rise at a fast rate, given the technological coverage observed within the last ten years. Naturally, the growth of networks ensued, promoting a rising concern regarding security [61]. In this scenario, it is expected that unusual traffic in the networks could mean that a device is being exploited. Another example is concerned with credit card fraud, where anomalies in a credit card report could indicate fraudulent activities associated with it. In the case of an EEG scan, anomalies in the frequencies could indicate a seizure and/or its pre/post conditions. An MRI image with visual anomalies can indicate the presence of a tumor.

Anomalies can be extrapolated from a wide range of areas. With this in mind, [5] categorize them in the following groups:

Point Anomaly: When a particular data point deviates from the normal pattern seen in the dataset. For example, when a person's normal car fuel usage is 5 liters per day, but for a particular day it becomes five hundred liters, that is a point anomaly.

Contextual Anomaly: When a data instance describes anomalously in a particular environment, it is called contextual or conditional anomaly. For example, it is usual for a person to spend more money during Christmas and new year season than for the rest of the year. Although spent money can be high, it may not be anomalous as high expenses are common during these periods. On the other hand, an equally high expenditure during a non-festive season could be deemed a contextual anomaly.

Collective Anomaly: When a collection of similar data instances behave erratically with respect to the entire dataset, the group of data instances is called a collective anomaly. In a EEG scan, the existence of low values for a long period indicates an outlying phenomenon translating to an abnormal premature contraction, whereas one low value by itself is not anomalous.

The following subsections present state of the art approaches on anomaly detection. A brief description and application are provided for each of them.

2.2.1 Classification-Based

Classification-based techniques rely on experts' knowledge on anomaly features. When an expert provides an attack's details to the detection system, an attack with a known pattern can be detected as soon as it is launched. For Intrusion Detection Systems (IDSs), this is usually deemed the signature-based detection method. This type of system can only detect attacks that have been previously learned by means of a signature. Even if a new attack's signature is introduced into the system, the effects of the first approach cannot be undone. The advantage of classification-based methods lies in their capability to detect novel attacks, if these present an ample enough deviation. However, given the lack of sufficient data to form a normal profile from which to learn, the system is prone to a high false classification rate [4]. In addition, these techniques require an expert. Next, follows a description and use cases of classification-based techniques.

Support Vector Machine

As seen in section 2.1.2, Support Vector Machine (SVM) are a special class of machine learning algorithms that perform classification through projection of the data instances in a dimension where separation of classes is achieved through a decision hyperplane. SVMs are also frequent in the domain of anomaly detection.

In [31], a framework on unsupervised intrusion anomaly detection is presented. The algorithms in use by the framework include k-NN as well as One-Class Support Vector Machine (OCSVM). The implemented OCSVM uses the RBF kernel for feature space mapping. SVM is a supervised learning algorithm by nature. In this use case, the data is unlabeled, so the SVM works differently:

- Instead of separating one class from another, it tries to separate the data points from the origin. This is possible through a quadratic program solution that penalizes points close to the origin while simultaneously trying to maximize the distance of the hyperplane from the origin;
- Support vectors also computed differently. The algorithm attempts to find a region where most of the training data lies and labels points in that region as class $+1$, while points in other regions are labeled as -1.

Overall, results show that for all considered methods, the detection rate was acceptable, considering how the threshold also affects the false positive rate. In any case, one has to account for the difficulty in unsupervised anomaly detection, given that the training data has no label and may not be clean.

In [37], a Host-based Intrusion Detection System (IDS) that monitors accesses to Windows Registry using Registry Anomaly Detection (RAD) is presented. This IDS uses a OCSVM trained on a dataset of normal registry accesses and is compared with a Probabilistic Anomaly Detection (PAD) algorithm. The implemented OCSVM was modified to compute kernel entries dynamically, given memory limitations. A total of three kernels were tested: linear, polynomial and gaussian. This implementation also uses Sequential Minimal Optimization (SMO). Results show that, overall, the linear kernel helped produce the best ROC curve, considering all tested kernels, but the PAD algorithm consistently presented better results than the OCSVM. The authors claim the PAD algorithm's advantage lies in the use of a Dirichlet-based hierarchical prior to estimate probabilities.

In [41], the performance of Robust Support Vector Machines (RSVMs) was compared with conventional SVMs and nearest neighbor classifiers, in separating anomalies from regular usage profile in computer programs. The authors present a new approach based on RSVMs that deals with noise in the dataset and thus exacerbates RSVMs' superiority in generalization capability over noisy data. Moreover, the number of support vectors in use for the RSVM (30 for clean data, 15 for noisy data) is significantly lower than for conventional SVMs (45 for clean data, 40 for noisy data), thus promoting a shorter execution time. Results show promise for RSVM, with this algorithm scoring the highest attack detection rate, while minimizing the false positive rate, in comparison with other algorithms in use, for both clean and noisy data.

Long Short Term Memory Recurrent Neural Network

As explained in section 2.1.2, LSTMs are Recurrent Neural Networks (RNNs) that bind memories and time input, and have also been recently explored in regards to anomaly detection.

Audio anomaly detection is addressed through the use of a purely unsupervised and novel approach in [60]. The implemented algorithms consist of autoencoders (basic, compressed, denoising and non-linear predictive) built on RNNs (both conventional and bidirectional) with LSTM memory blocks. The performance evaluation is weighted on datasets detailing normal behavior (characterized by sounds from home environments) and assessed through the F-score metric. Results show that all considered autoencoders show promise, with the lowest scoring implementation (LSTM-Compression Auto Enconder) at 89.1%. The best configuration was the Non-Linear Predictive Bidirectional LSTM Denoising Auto Encoder, scoring at 94.4%.

Another application of Anomaly Detection through the use of LSTM is present in [24], where a predictive model for healthy ECG signals is proposed. This model consists of a deep LSTM neural network where the memory units are stacked in order to feed forward to the following unit. Experiments on this model consist of training the models on the MIT-BIH Arrhythmia Database, which contains both normal and irregular behavior. Irregular behavior is categorized into four different types of beats that should be discriminated. It is worth noting that the authors highlight a related objective of the paper "*that the deep LSTM generalize to multiple anomaly* *types (Arrhythmias)*". Results show that the model obtains an F1-score above 93% for all of the irregularities except Atrial Premature Contraction, which scores 85%.

Bayesian Network

The Bayesian Network (BN), also known as belief network (or Bayes net) is a directed acyclic graph in wich each edge corresponds to a conditional dependency between unique random variables (represented as graph nodes). BNs belong to the family of probabilistic graphical models [14] and use Bayesian inference for probability computations.

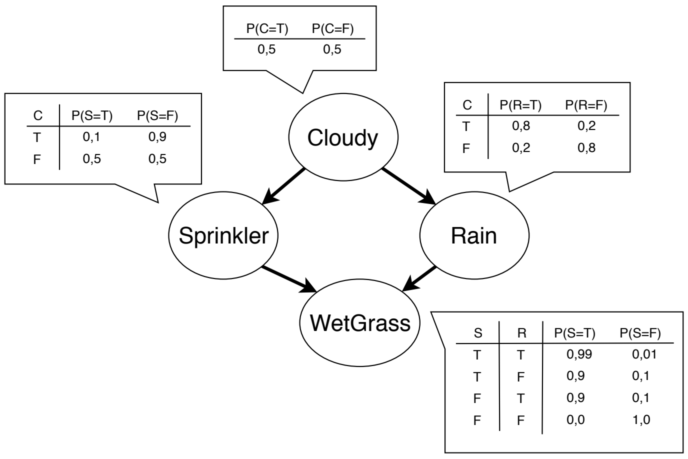


Figure 2.16: Bayesian network with highlighted joint probability distributions attached to each random variable. Based on [82].

If an edge (A,B) exists in the graph connecting random variables A and B, $P(B|A)$ is a factor in the joint probability distribution, so we must know $P(B|A)$ in order to conduct inference for all values of B and A.

Anomaly based approaches can detect previously unknown attacks, but suffer from difficulty in building a robust model which may result in a large number of false positives. In [49], the authors identify two main reasons for this proportion: simplistic aggregation of model outputs in the decision phase and lack of integration of additional information into the decision process. To deal with these shortcomings, an event classification scheme based on Bayesian Networks is proposed, in the context of intrusion detection on system calls. The integrated models in the net verify string length, character distribution, structure and tokens. The experiment includes a comparison with a naive threshold-based approach, and results show that the BN significantly reduces the false positives throughout the ROC curve. In addition, when all attacks are detected, only half as many false positives in the threshold-based approach are registered.

2.2.2 Statistical Anomaly Detection

Statistical theories are also useful when modeling intrusion detection systems. For example, the popular chi-square theory is used for anomaly detection [90]. In this technique, anomaly detection derives from events that are sufficiently deviated from normal. A distance measure based on the chi-square test statistic is developed as

$$X^2 = \sum_{i=1}^{n} \frac{(X_i - E_i)^2}{E_i}, \tag{2.31}$$

where X_i is the observed value, E_i is the expected value and n is the number of variables. X^2 has a low value when an observation is near the expected. Following the $\mu \pm 3\sigma$ rule, when an observation, X^2, is greater than $\overline{X^2} + 3S_X^2$ it is considered an anomaly.

Next, follows a description and use cases of statistical techniques.

Principal Component Analysis

Principal Component Analysis (PCA) is a technique most used in the pattern recognition pipeline for feature reduction. The central idea for PCA is to reduce data dimensionality while retaining as much information as possible in the original dataset. This is made by transforming the data to a new set of variables, principal components(PCs), which possess the following properties, according to [44]:

- uncorrelated;
- orthogonal;
- their variances are sorted from highest to lowest or their total variance is equal to the variance in the original data.

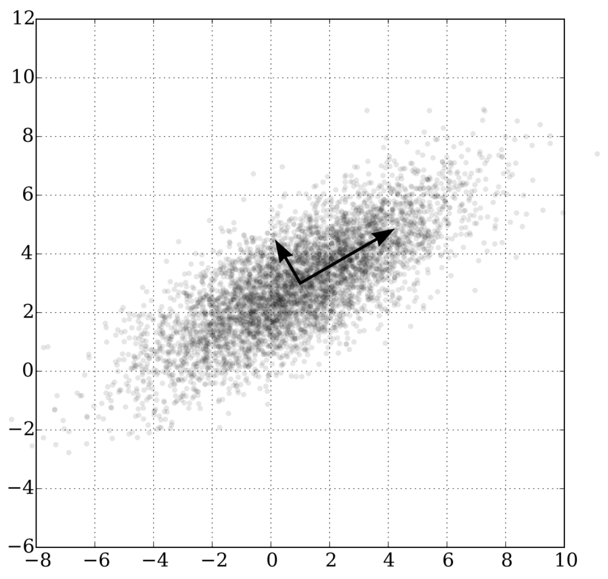


Figure 2.17: PCA applied in dataset. The first 2 PCs' directions are displayed.

To better understand how the PCs are obtained, one should look for the definition of key terms in PCA. The **variance** is a measure of variability that gives information on how far the data is spread. Its value is given by the squared deviation from the mean.

$$Var(X) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n} \tag{2.32}$$

Covariance is a measure of the joint variability of two random variables. Its value is computed as shown below.

$$Cov(X,Y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n} \tag{2.33}$$

Assuming a matrix A, almost all vectors change direction, when multiplied by A. An **eigenvector** is in the same direction as Ax. Multiply an eigenvector by A, and the vector Ax is a number λ times the original x. The number λ is an **eigenvalue** of A [84].

In PCA, the first step is to compute the covariance matrix of the data points, from which to extract the eigenvectors. At this point, the eigenvectors are sorted in descending order, by their eigenvalues. Depending on the number of PCs in use, the first k eigenvectors will be the new k dimensions. Finally, the original n dimensional data is transformed into k dimensions.

Many statistical based intrusion detection methods assume a normal distribution of the data or resort to the central limit theorem, requiring the number of features in the data to be greater than 30. In this case, it is usual for the data on these problems to present a high dimensionality. While PCA has found its use in feature reduction, it came to find its use in anomaly detection, in [80], where a novel scheme that uses robust PCA for intrusion detection in unsupervised data is proposed. Robustness on this approach is attained through multivariate trimming on the correlation matrix of the training data (preferably free of any outlier data points to begin with).

Adding to the uniqueness of the approach, the authors implement two functions of principal component scores, one for the major and another for the minor components. The second one is present to assist classify observations that do not conform to the normal correlation structure.

The number of components in use is determined by the variation in the data that is accounted for by the PCs. In the experiments, for one, a comparison on the number of considered PCs is carried out. Next, the authors compare the best configuration of the PCC to the Canberra metric, density based local outliers (LOF) and k-NN, with $k = 1$ and $k = 5$, through the KDD'99 dataset. Evaluation of the approaches is obtained through ROC curves plotted by variation of the false alarm rate threshold.

For the comparisons drawn out between different setups of the PCC (regarding variation accounting by the PCs), the PCC based on the major components that can explain 50% of the variation is the best, for a low false alarm rate, but also presents adequate results for higher rates.

When compared with other classifiers, the PCC stands out, with a near 99% consistent detection rate on all false alarm ratings, followed by 1-NN, but by a large margin.

Mixture Model

It is usual to refer to probabilistic models which take the form of simple distributions (gaussian, for example) in order to better analyze a data set. However, the data in use is not always simple. For example, it can be multimodal. That is, it contains several modes (regions of high probability mass) in between. Suppose we have collected temperature data from the shore and a mountainous region, but did not label the readings for the region in place. Plotting the collected data might show the distribution has at least two modes, as seen in figure 2.18.

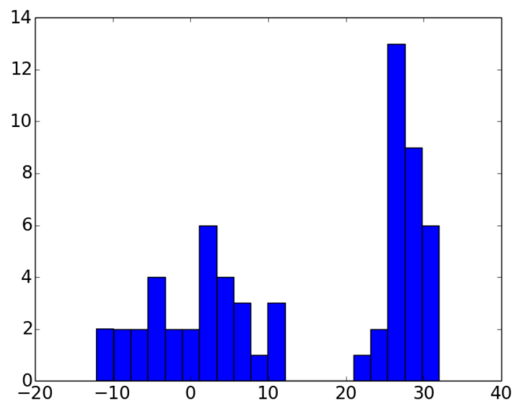


Figure 2.18: Histogram of daily high temperatures in °C.

In this case, one might model the data in terms of a mixture of components, where each component has a simple parametric form (such as a Gaussian). In other words, it is assumed each data point belongs to one of the components, and we try to infer to infer the distribution for each component. The **mixture model** is formally defined in terms of latent variables, which are never observed, usually denoted z. Variables which are always, or even sometimes, observed, are called observables

[36]. In the previous example, the region is the latent variable while the the temperature is the observable. In mixture models, the latent variable corresponds to the mixture component, and it takes values in a discrete set. In general, a mixture model assumes the data is generated through the following process:

1. Sample z
2. Sample the observables x from a distribution dependent on z

, i.e.

$$p(z, x) = p(z)p(x|z) \tag{2.34}$$

, with $p(z)$ being a multinomial distribution, $p(x|z)$ can take a variety of parametric forms, but for the example at hand, we will assume it is a gaussian distribution. Thus, this model is dubbed a mixture of Gaussians, as seen in figure 2.19.

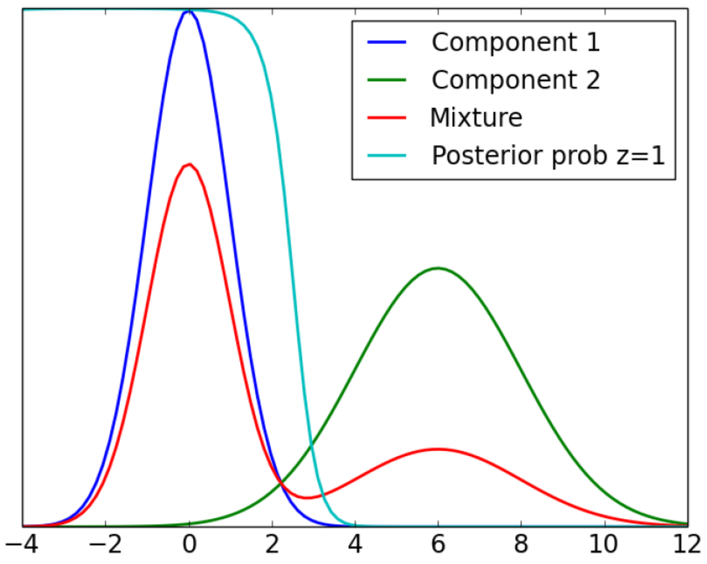


Figure 2.19: Example of a mixture of Gaussians model.

In [30], a mixture model is used for extrapolation on anomaly presence in system call data. Their approach for anomaly detection uses machine learning techniques to estimate a probability distribution and then applies a statistical test in order to detect anomalies. The data is modeled through the following generative distribution.

$$D = (1 - \lambda)M + \lambda A \tag{2.35}$$

, with M being a structured probability distribution estimated over the data through a machine learning technique. A is a distribution modeled on anomalous elements. Detection of the anomalies is attained by computation of the likelihood for each distribution. Performance comparison of their method is carried out in relation to the approaches `stide` and `t-stide`, based on their results in [87]. Results are presented in ROC curves and show that the presented method outperforms the baseline methods, when trained over noisy data, and performs comparably when trained with noisy data and compared with the baseline methods trained over clean data (as in, there are no anomaly data instances). It is important to note that the presented approach makes relevant assumptions regarding the data: normal data can effectively be modeled using the probability distribution; anomalous instances are sufficiently different from normal ones, in order to be correctly detected; the number of anomalies is considerably small compared to the number of normal instances (comparisons are drawn out only over program traces where the percentage of intrusions is less than 5%).

2.2.3 Information Theory

Information theoretic methods compute information measures of a dataset. These methods assume that anomalies significantly alter the information content of the otherwise "normal" dataset. Usually, metrics are calculated using the whole dataset and then the sequences of points whose elimination induces the biggest change in the metric are labeled anomalous. In [53], information theory measures (entropy, conditional entropy, relative conditional entropy and information gain and cost) are used to characterize several case studies in the domain of internet security.

Entropy measures the uncertainty of a collection of data items. For a dataset X where each data item belongs to a class $x \in Cx$, the entropy of X relative to this $|C_X|$-wise classification is defined as

$$H(X) = \sum_{x \in C_X} P(x) log \frac{1}{P(x)}, \tag{2.36}$$

where $P(x)$ is the probability of x in X.

Usually, in Information Theory, entropy is regarded as specifying the number of bits required to encode the classification of a data item. The value is smaller when the class distribution is skewer. For example, if all data items belong to the same class, then entropy is 0, because there is only one possible outcome out of the classification.

In anomaly detection, entropy can be used to measure the regularity of the audit data [53]. Each unique record in an audit dataset represents a class. The smaller the entropy, the fewer the number of distinct records. Thus, the audit dataset is more regular.

The **conditional entropy** of X given Y is the entropy of the probability distribution $P(x|y)$, such that,

$$H(X|Y) = \sum_{x,y \in C_X, C_Y} P(x,y) log \frac{1}{P(x|y)}, \tag{2.37}$$

where $P(x,y)$ is the joint probability of x and y and $P(x|y)$ is the conditional probability of x given y.

The **relative entropy** between two probability distributions $p(x)$ and $q(x)$ that are defined over the same $x \in C_X$ is

$$relEntropy(p|q) = \sum_{x \in C_X} p(x) log \frac{p(x)}{q(x)} \tag{2.38}$$

The relative conditional entropy between two probability distributions $p(x|y)$ and $q(x|y)$ that are defined over the same $x \in C_X$ and $y \in C_Y$ is

$$relCondEntropy = \sum_{x,y \in C_X, C_Y} p(x,y) log \frac{p(x|y)}{q(x|y)} \tag{2.39}$$

Based on the previously presented information measures, appropriate anomaly detection models can be built, as shown in [53].

2.2.4 Clustering-Based

Clustering entails grouping similar data points within a small area (a cluster), while those that appear different are either in another cluster of similar points, or simply do not belong to any formed cluster [4]. This approach is primarily an unsupervised technique, although semi-supervised clustering has been explored [12]. Although there are different types of clustering techniques, the focus in this documents befalls on regular and co-clustering. There are three key assumptions to make when using clustering algorithms for anomaly detection which are presented below.

Assumption 1: normal data instances belong to a cluster, while anomalies belong to any other cluster they form.

Assumption 2: normal data instances are close to their cluster centroid. Anomalies are far away from any normal cluster centroid.

Assumption 3: normal data instances belong to large and dense clusters, while anomalies belong to small and sparse clusters.

Regular Clustering

Regular clustering techniques cluster the data considering the rows of the dataset. One popular example of this variant is k-means clustering [66].

In k-means, the goal is to find k groups in the data. Iteratively, the algorithm assigns each data point to one of the k groups based on the features that are provided. In the end, the algorithm outputs the centroids of the k clusters, which can be used to label new data and labels for the training data (whose points are assigned to their respective cluster).

Once the clusters are built, new data points are subject to the following assumptions.

- An instance is classified as normal, if it is closer to the normal cluster centroid than to any anomalous one. The reverse holds for an anomalous data instance;
- If the distance between the data instance and any normal cluster centroid is larger than a pre-defined threshold, the instance is deemed anomalous.

As seen for SVMs, in [31], a framework on unsupervised intrusion anomaly detection is presented. Other than the OCSVM, both k-NN and a Cluster-based estimation algorithm are experimented with. The cluster-based estimation's goal is to compute how many points are near each point in the feature space. The distance threshold is defined manually. Complexity on this approach is high (O(n^2)), given the pairwise distance measures between all points. To overcome this, the authors perform fixed-width clustering on the entire training data set. For each new point, if it falls outside of the fixed-width of the cluster, it becomes the new centroid of a new cluster. A similar problem is faced and dealt with in the deployment of k-NN. The authors employ canopy clustering [62] in order to compute the k-nearest points to a given point, which significantly reduces the algorithm's execution time while promoting real-time application in IDS systems. As previously shown, results for all methods present acceptable results, making the use of clustering algorithms a promising choice in anomaly detection.

In [66], the first part of the paper focuses on Network Data Mining (NDM), while in the second part, a flow-based anomaly detection scheme based on k-means clustering is proposed. The NDM approach in use deploys k-means in order to separate time intervals with normal and abnormal traffic in the training dataset. The resulting centroids are used for anomaly detection in new unseen data.

It should be noted that the authors employ k-means not only for classification of the data points, but also for outlier detection. A unseen data point may be classified as anomalous if the distance to the abnormal centroid is smaller than the distance to the normal centroid. It is an outlier, and, therefore, an anomaly, if the distance from the normal centroid exceeds a defined threshold. Experimental results on this model are based on both data generated from a local testbed, and `tcpdump` traces from a real network. The testbed experiments enables an assertion on the basic anomaly detection capabilities of the proposed model. With the real traffic traces, an analysis on the UDP traffic revealed a short burst (of 15 to 20 seconds on the original time scale) which the authors account to as being the result of the client program of a game updating its list of available game servers. A more sophisticated performance metric has not been provided.

Co-Clustering

Co-clustering is similar to regular clustering, but instead of processing the dataset for rows, it processes rows and columns simultaneously. It can produce a set of c column clusters of the original columns (C) and a set of r row clusters of the original row instances (R). Co-clustering defines and optimizes a clustering criterion to find subsets of rows and columns of a data matrix [4].

Advantages of this variation over Regular clustering include:

- Simultaneous row and column grouping provides a more compact representation of the data while preserving information;
- Co-clustering can be considered a dimensionality reduction technique and is suitable for creating new features;
- Provides significant reduction in computational complexity. Traditional k-means has $O(mnk)$, where m is the number of rows, n is the number of columns and k is the number of clusters. In co-clustering, computational complexity is $O(mkl+nkl)$, where l is the number of column clusters. It is clear that $O(mnk) > O(mkl + nkl)$.

In [3], an adaptation of co-clustering is developed in order to analyze network traffic patterns, in regards to DoS attacks as a collective anomaly. The authors extend co-clustering by enabling the algorithm to handle mixed attribute data instances, which translates to an increment in detection accuracy. They also claim to be the first to explore collective anomaly detection, only drawing comparison of the results with [72], who used co-clustering for network anomaly detection considering only cluster purity as a performance measure, and only seven numerical attributes for the clustering algorithm. In any case, the former manages to outperform the latter's implementation. The obtained F-score reaches 96%.

2.3 Case-based Reasoning

As defined in [47], Case-based Reasoning (CBR) is a problem solving paradigm that attempts to replicate human behavior towards reasoning and learning, from a psychological perspective. It differentiates itself from other AI approaches by relying on specific knowledge instead of a generalization within a problem domain. To solve emerging problems (new cases), CBR draws comparisons between previously solved, similar cases (which have been stored), and the new case, applying the same solution to the latter. Once the new problem too is solved, it is added to this storage, commonly known as Case-Base. This process is best described in the CBR cycle.

2.3.1 The CBR cycle

The CBR cycle details the processes in use when applying CBR, and is described by four stages, present in Figure 2.20.

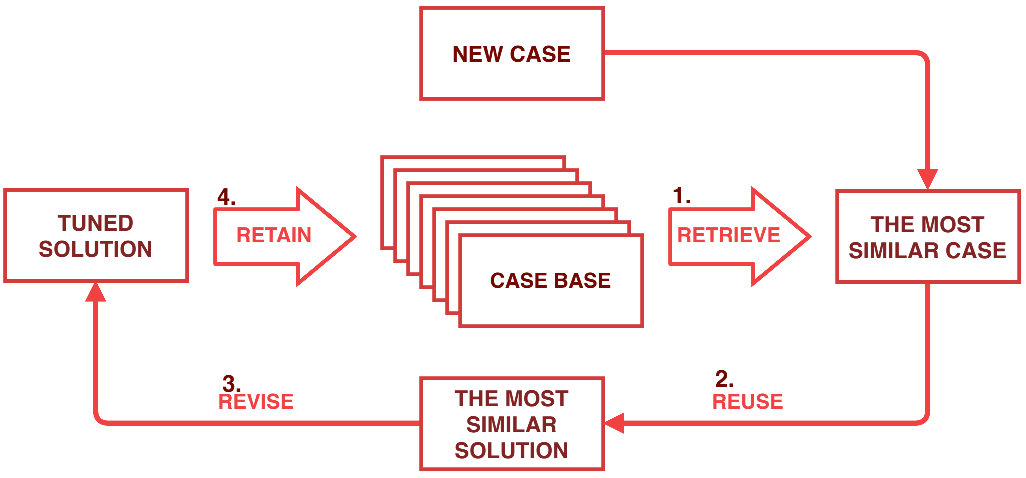


Figure 2.20: Case-based reasoning cycle.

Retrieve: Given a new case, it is compared with cases in the case base. The most similar case is then retrieved.

Reuse: The previous similar cases' solution is mapped into a new solution, for the current problem.

Revise: Unless the new and previous case's solution is an exact copy, it is susceptible to adaptation and, therefor, revisions and tests.

Retain: If the solution is successfully reformulated to the target case, this case is retained and added to the case base.

2.3.2 Case Base

The storage of the cases is an important aspect while developing a CBR system. It should provide the means to create an efficient and accurate search. However, accuracy and efficiency are qualities that are inversely proportional most of the times [71]. The decision for the best solution is domain specific and weights on a compromise between accuracy and efficiency. In general, the three main considered approaches are:

Flat Organization: The case base follows a flat structure, such as a list, which is simple to create and maintain. Search is usually done in a case-by-case search, promoting accuracy, but severely impacting efficiency.

Clustered Organization: The case base aggregates similar cases into clusters. The advantage is that it can reduce the time needed to find the best match since they are grouped by similarity. The disadvantage is that it needs a complex algorithm to add and delete cases, which makes it harder to maintain. The quality of the cluster may also reduce the accuracy of the system.

Hierarchical Organization: When cases share the same features and the domain allows categories, these cases can be grouped together. Each case has an assigned category and its category is inter-linked within the structure. It has the disadvantage of higher complexity but the advantage of fast and accurate case retrieval.

2.3.3 Similarity Measure

A retrieval algorithm should return cases that are the most similar to the target problem. However, choosing the best matching cases can be a complex task. There are many ways of retrieving cases, including nearest neighbor search, serial search, hierarchical search, parallel search, and others [26]. Still, the quality of the matched cases weights heavily on the similarity measure. Usually, the measure is domain specific and requires a domain expert, but it is possible to apply direct algorithms and methods on the cases' feature vector, such as an euclidean distance.

2.3.4 Adaptation

As previously mentioned, once a matching case is chosen, it may not correspond exactly to the same problem and may need adaptation. While this is a domain specific issue, there are a few general approaches to deal with adaptation [77, 89]:

Structural Adaptation: Applies rules directly to the solution stored in cases. Can include modifying certain parameters and interpolate various cases to generate a good solution. This requires knowledge on the problem domain.

Abstraction Adaptation: Should a fraction of the matched solution not be applicable to the target problem, it is abstracted and then specified, if possible.

Critic-based Adaptation: A solution is proposed and its features are checked for problems. Each problem has a pre-defined strategy to solve it. Side effects of each strategy must also be taken into consideration.

Derivational Adaptation: Reuses the methods that contributed to the original solution to produce a new one. This requires knowing the planning sequence that produces a solution. This adaptation is highly domain specific and can only be applied in well understood problem domains. This includes replacing a step in the solution generator to contextualize it in the problem.

2.4 Bayesian Optimization

In [64], a global optimization problem can be described as any decision problem (with an objective function f to be minimized or maximized), if there is no additional information indicating that there is only one minimum, or maximum. The global optimization problem is usually dismissed by statements such as "*If there are many minima (or maxima), they should all be compared.*" This statement is correct if it is a practical possibility to:

- find any finite number of local minima;
- prove that no more local minima exist.

which is possible in just a few, special cases. Generally, the interest lies not in finding an exact global minimum, but rather focusing on finding approximations to it. In this case, the main problem is how to define the approximate solutions and how to find them.

Bayesian Optimization is an option for global optimization of black-box functions, where samples are limited due to their expense [20]. It iteratively develops a global statistical model (often called surrogate model) of the unknown objective function. It starts with a prior over the space of possible objectives in f and a likelihood, to get a posterior measure over the objective, given some observations, followed by an acquisition function used to map beliefs about f to a measure of how promising each instance in the input space is, if it were to be evaluated next. The goal is finding an input that maximizes the acquisition function and submit it for function evaluation. An acquisition function should address the exploitation vs. exploration trade-off, that is, the idea that we are interested both in regions where the model believes the objective function is low (exploitation), and regions where uncertainty is high (exploration). Usually, this acquisition function is the Expected Improvement (EI) [65], defined as the expected amount of improvement over some target t, if we were to evaluate the objective function at x. EI encourages both exploitation and exploration because it is large for inputs with a low predictive mean (exploitation) and/or high predictive variance (exploration).

Take as an example the scenario where we have a regression problem with an LSTM being used and we want to find the optimal number of hidden layers, with sMAPE being the evaluation metric. Therefore, our output, $f(x)$, is the sMAPE, and our variable, x, is the number of hidden layers.

First, we define a surrogate model, in this case, Gaussian Process (GP), as seen in figure 2.21.

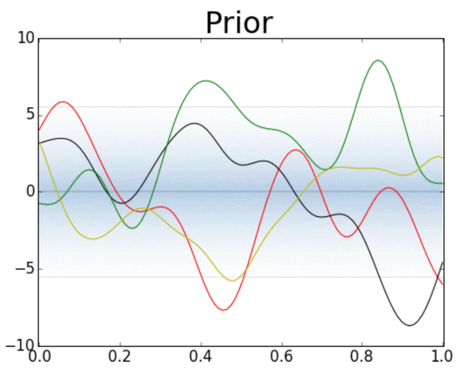


Figure 2.21: GP showing the possible search space. Based on [34].

In the following step, we combine the prior and the likelihood to obtain a posterior over the objective, given an observation. Figure 2.22 refers to posteriors drawn from four observations.

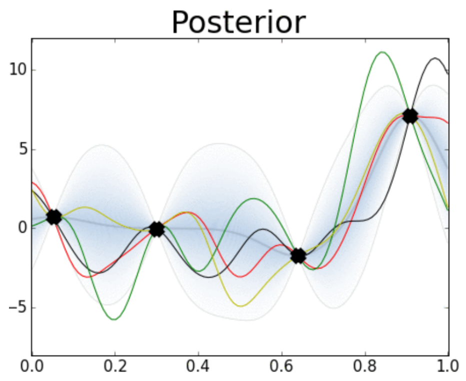
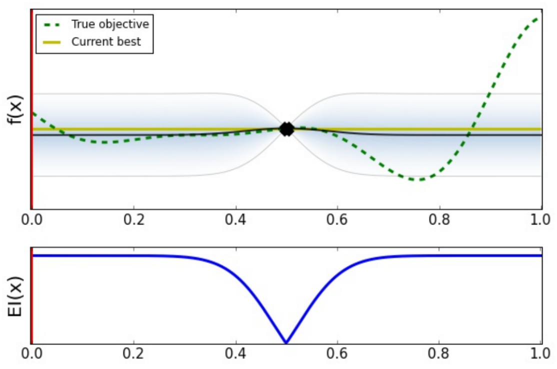
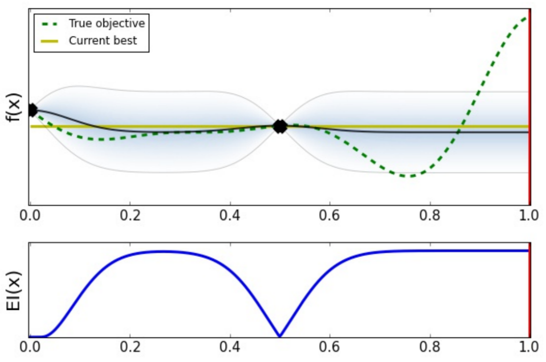


Figure 2.22: GP after being given observations from the true objective function. Note how the search space is formatted in accordance to predictions performed by the model. Based on [34].

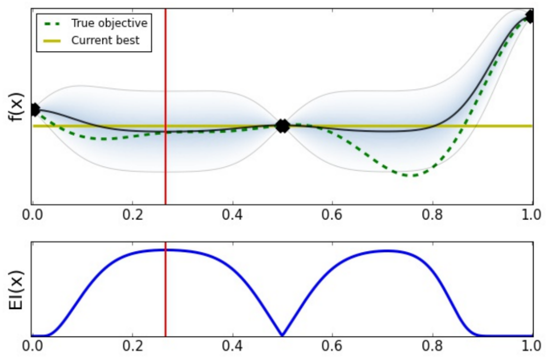
Next, we apply the posterior to an utility function to decide where to evaluate next. This step is followed by a data update. Figure 2.23 shows a few iterations.



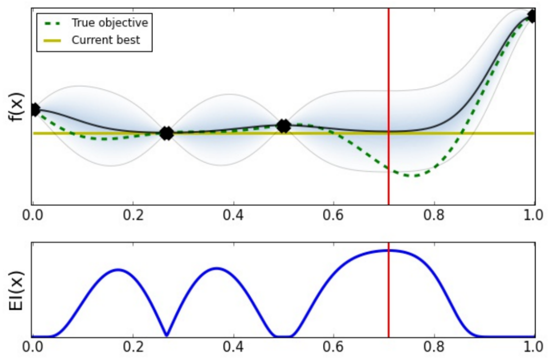
(a) Iteration 1.



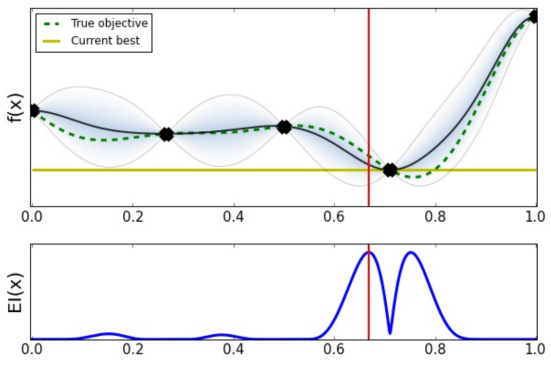
(b) Iteration 2.



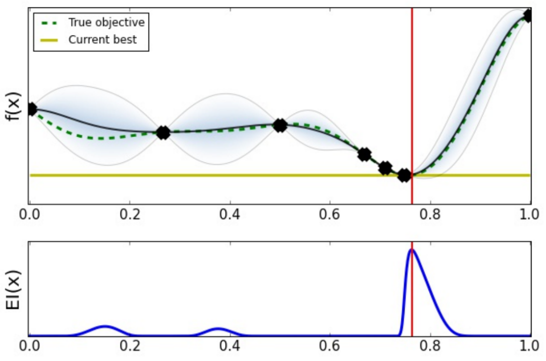
(c) Iteration 3.



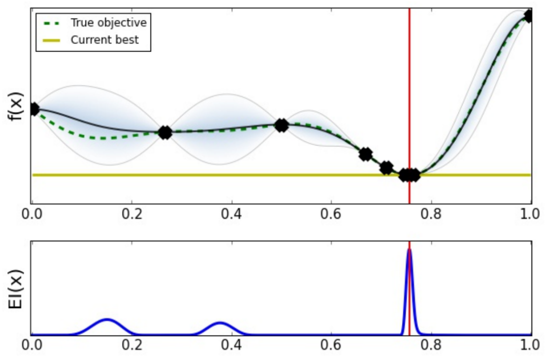
(d) Iteration 4.



(e) Iteration 5.



(f) Iteration 7.



(g) Iteration 8.

Figure 2.23: Iterative process within Bayesian Optimization applied on a global minimum optimization. Based on [34].

From here, we can see that a good approximation on the global minimum has been obtained. A LSTM where the number of hidden layers is close to x would present the lowest registered Symmetric Mean Absolute Percentage Error (sMAPE).

2.5 Competitor Analysis

Time series span throughout a wide range of areas, from financial information, to meteorological readings, to IoT data. It is only natural that to better understand trendings, anomaly occurrences and forecasting, one has to perform analysis on these data structures. This procedure can be facilitated with the help of frameworks which provide the analyst with tools to perform forecasting and anomaly detection. The following section addresses competitors to time series analysis tools and how they are related, or not, to each other. Both forecasting and anomaly detection applications are considered. Finally, comparisons between each competitor are summarized.

2.5.1 Forecasting

NCSS 12

NCSS is a company created in 1981 by Jerry L. Hintze that specializes in statistical software applied to medical and business research, quality control, academia and teaching/learning. The latest iteration on their data analysis product, NCSS 12[1], has been released in November, 2018 and aggregates a rich library on statistic procedures (including descriptive statistics, parametric and nonparametric tests, forecasting, multivariate analysis, time series and model fitting). Operations on this platform appear intuitive and clear: one may import and format the initial data through a table structure; open a procedure which they want to apply; and view the output on another window. Major data extensions (such as MATLAB, SPSS, text, R, Excel and more) are also supported, making this suite's data management flexible. Procedure selection is also presented in a straight-forward manner, through a dropdown menu, search bar and even a category tree. Output handling is an easy task, as direct interaction with the plots highlights each one of them, should the user want to isolate one in a separate file.

RapidMiner Studio

RapidMiner Studio[2] is a visual data science workflow designer accelerating the prototyping and validation of models. Data access and management is flexible, as the application supports both traditional structured data (time series) and unstructured data (text, images, and media). Information extraction from these types and transformation from unstructured to structured data is also attainable. Data exploration is also delivered, with the system being able to compute descriptive statistics and to render graphical information. Data preparation is available through the use of processes such as sampling, transformations, partitioning, binning, weighting and selection. The selection of offered models is rich, including both supervised and unsupervised learning. Both classical and machine learning (including deep learning) approaches are available. Validation on the previous models can be performed on a set of provided performance measures. The characteristic the company advertises for standing out from the competition is the abstraction drawn from actual programming of the previously mentioned data analysis pipeline. The company boasts a set of utility process control operations that enables process construction that behave like a program over which one can iterate tasks, branch flows and system resource calls.

[1]`https://www.ncss.com/software/ncss/`
[2]`https://rapidminer.com/products/studio/feature-list/`

Autobox

Automatic Forecasting Systems Inc. was founded in 1975 with the goal of presenting the first marketable forecasting toolset. At AFS, two central beliefs are presented: the Box-Jenkins approach provides the proper framework for forecasting; procedures that consist of methods applied in a consistent way are subject to automation. Thus, they developed a forecasting engine that applies the modeling philosophy of Box-Jenkins to time series and builds models automatically. The result is a package that provides a set of tools for both beginners and experts, Autobox[3]. Autobox won the "Best Dedicated Forecasting Program" in [9]. Major features include adaptation of the model to the data through causal variable discovery by patterns from historical forecast errors and outliers identified by the Autobox engine. Standard features include outlier classification in four areas and forecasting diagnostics through ARIMA and Transfer Function models.

Alteryx Designer

Formerly known as SRC, LLC, founded in 1997, Alteryx is a software company specialized in data science and analytics. Their product, Alteryx Designer[4], streamlines the process of data preparation, blending and analytics by delivering a repeatable workflow for self-service data analytics, leading to deeper insights within an acceptable time window, all while maintaining an intuitive user interface. Alteryx promises data preparation and blending capabilities that are up to a hundred times faster than traditional approaches. Data access is done through data warehouses and cloud applications as well as local sources, such as spreadsheets. With the use of Alteryx Visualytics, changes in the preparation, blending and modelling stages can be interpreted in a graphical manner. Prediction analysis makes use of R-based analytics while removing the coding layer from the end-user, by option. Given the recent popularity of Big Data, IoT, mobile consumer and social media data, spatial analytics are embedded in this software. Finally, insights sharing is simplified in the form of graphics exportation to popular reporting formats (PNG, HTML, PDF and Microsoft Office extensions).

Microsoft Azure Data Explorer

Microsoft's Azure Data Explorer[5] was published by Microsoft and presents a fast and scalable data exploration service for log and telemetry data. It is a good option for analyzing large volumes of data stemming from websites, applications and IoT devices. This tool mostly focuses on quickly identifying trends, patterns or anomalies in all data types inclusive of structured, semi-structured and unstructured data, through a powerful query language optimized for ad-hoc data exploration.

Amazon Forecast

Amazon stands out globally for being the most valuable company in world[6]. This company focuses in diverse sectors, such as e-commerce, cloud computing and artificial intelligence. It comes as no surprise that they hold a product capable of Machine Learning (ML) induced forecasting. Amazon Forecast[7] is an accurate time-series forecasting service that depends on ML to deliver accurate forecasts. Accuracy on the forecasts is promoted on time series data as well as additional variables that have influence. Machine learning expertise is also not an issue as this service includes AutoML capabilities. Once the data is uploaded, Amazon Forecast processes the data, selects the appropriate algorithms, trains a model, provides registered metrics and generates forecasts. If conformant with Amazon Forecast requirements, consumers can provide additional algorithms to

[3]`https://autobox.com/cms/index.php/products/autobox`

[4]`https://www.alteryx.com/products/alteryx-platform/alteryx-designer`

[5]`https://docs.microsoft.com/en-us/azure/data-explorer/time-series-analysis`

[6]`https://edition.cnn.com/2019/01/08/investing/amazon-most-valuable-company-microsoft-google-apple/index.html`

[7]`https://aws.amazon.com/forecast`

the platform and even customize existing ones. Finally, forecasts can be visualized in the console, exported in batch in `CSV` format or be retrieved using the Amazon Forecast API.

Forecasting Competitors Analysis

It is clear that, while Autobox has been one of the first publicly available products, it is severely outperformed by its competitors. NCSS, RapidMiner and Alteryx Designer present the software suites with the richest set of features, while Microsoft's Azure Data Explorer is not entirely a consumer ready product, but a powerful service to be used majorly by experts in the domain of computer science and statistics, given it's optimized query language. Amazon Forecast takes a different approach, presenting a cloud solution. However, its AutoML capabilities render it an appealing choice, for consumers without ML expertise. In Table 2.1 lies a summary on forecasting competitors' features.

Table 2.1: Forecasting competitor feature comparison.

	Competitors					
Features	**NCSS**	**RapidMiner**	**Autobox**	**Alteryx Designer**	**MS Azure Data Explorer**	**Amazon Forecast**
Auto ML		●				●
Semi-structured data	●	●		●	●	
Unstructured data	●	●		●	●	
Descriptive Statistics	●	●	●	●	●	●
Data Visualization	●	●	●	●	●	●
Data Processing	●	●	●	●	●	●
Supervised Learning	●	●		●		●
Unsupervised Learning	●	●		●		●

2.5.2 Anomaly Detection

Microsoft Azure Machine Learning Studio

Another ambitious service provided by Microsoft is the Azure Machine Learning Studio[8]. This collaborative visual development environment helps build, test and deploy predictive analytics solutions in the cloud. The Machine Learning Studio contains a module specifically designed for time series data processing, namely in anomaly detection, but one can also take advantage of the R and Python scripts in use to manually manipulate the data, and apply it to the set of given classification and regression models. Classification modules range from binary to multiclass and include popular algorithms such as Decision Forest, Decision Jungle, Neural Network, Support Vector Machine (SVM), Decision Tree and classification models ensembles. Regarding Regression models, the platform includes bayesian linear, boosted decision tree, fast forest quantile, neural network, ordinal and poisson regressions.

Elastic Stack

Elastic Stack is a collection of open-source software that facilitates log search, analysis and visualization, commonly referred to as centralized logging. Centralized logging is useful when trying to identify problems, in servers and applications, and issues that span multiple servers by correlating their logs during a specific time window. Elastic Stack has four main modules:

Kibana: gives shape to data and is the extensible user interface for configuring and managing Elastic Stack;

Elasticsearch: distributed, JSON-based search and analytics engine;

Beats: platform for lightweight data shippers that send data from edge machines to Logstash and Elasticsearch;

Logstash: data processing component of Elastic Stack which sends incoming data to Elasticsearch.

Kibana[9] is a module that stands out for enabling time series analysis (including queries, transformations and visualizations) on its dedicated UI. Anomaly Detection is present and applied with the use of unsupervised ML features. The forecasting feature is also boasted, through ML techniques, but no clear list of implemented algorithms is provided, other than a mixture of techniques which include clustering, various types of time series decomposition, bayesian distribution modeling and correlation analysis.

Sophie AIOps

Loom Systems is a company founded in 2015 which created an ML powered log analysis software that provides real time solutions for IT incidents, Sophie AIOps[10]. Sophie parses structured and semi structured data, classifies them and then applies the appropriate measurement method for each property. Anomaly detection is supported by ML algorithms and optimized with dynamic thresholds based on data signature in real-time. Sophie applies cognitive reasoning to detect cross-environment and cross-application issues in order to analyse the root cause for the identified issues.

Anomaly Detection Competitors Analysis

Regarding Anomaly Detection, Microsoft's Azure Machine Learning Studio presents the software with the richest libraries in terms os algorithms in use, and even though information on Elastic

[8]`https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/time-series`

[9]`https://www.elastic.co/products/kibana`

[10]`https://www.loomsystems.com/product-overview`

Stack is lacking, it still is the only open-source solution at hand. Sophie AIOps heavily markets its cross-correlation capabilities in order to better understand identified anomalies.Table 2.2 summarizes anomaly detection competitors' features.

Table 2.2: Anomaly detection competitor feature comparison.

	Competitors		
Features	**MS Azure Machine Learning Studio**	**Elastic Stack**	**Sophie AIOps**
Auto ML		●	
Semi-structured data	●	●	●
Unstructured data	●	●	
Descriptive Statistics	●	●	●
Data Visualization	●	●	●
Data Processing	●	●	●
Supervised Learning	●		●
Unsupervised Learning	●	●	●

2.6 Final Remarks

Sections 2.1 and 2.2 of this chapter highlight forecasting and anomaly detection frameworks heavily based on some of the ML approaches presented in this State of the Art. In this section, a brief analysis on all of the aforementioned competitors is drawn.

The focus of the work included in the internship goes to the creation of a framework capable of processing time series in order to obtain forecasts or detect anomalies as accurately as possible. Given the use of ML approaches, the model choice and its hyperparameter tuning is of the utmost importance. This is where Auto ML comes into use, by simplifying a pipeline which would otherwise require more time to research, develop and deploy. Table 2.3 shows that, from all of the presented competitors, most seem to neglect this feature, apart from RapidMiner and Elastic Stack. Most of them also focus only on one type of analysis, which renders them a weaker offer. This table was inverted to better readability.

Table 2.3: Competitor comparison regarding time series analysis (in forecasting and anomaly detection) and Auto ML features.

	Services		
Competitors	**Forecasting**	**Anomaly Detection**	**Auto ML**
NCSS	●		
RapidMiner	●	●	●
Autobox	●		
Alteryx Designer	●		
MS Azure Data Explorer	●		
Amazon Forecast	●		
MS Azure ML Studio	●	●	
Elastic Stack	●	●	●
Sophie AIOps		●	

Chapter 3

Resources and Tools

This chapter details all the resources and tools to consider for the approach proposed in this report. The resources constitute a list of datasets from which to input into the models, while the tools include model implementation, data processing and other Machine Learning (ML) pipeline modules.

All resources and tools must be licensed under open-source licenses, such as: BSD License[1], Apache License[2], MIT License[3] and GPL[4], so that developers are free to use, modify and share library implementations.

3.1 Resources

In this section, some useful resources for the framework are presented. The datasets provided by the resources that follow serve as a good benchmark to consider when assessing the selected model's quality and performance, given that some of them even provide solutions in the forecasting front. These resources stem from renowned competitions and publicly available repositories. While the internship takes place at Novabase Business Solutions, it is worth mentioning a few considerations regarding the use of this data. The sector in which this internship unfolds is the Financial Solutions Industry, which specializes in solutions for banks, insurance agencies and capital markets. Given the data sensitivity, regarding privacy, provided in these areas, the usage of this data will not be considered as a viable solution for resources.

3.1.1 Makridakis Competition (M-Competition)

The M-Competitions represent one of the most worldwide popular forecasting time series competitions. The datasets provided for these competitions usually contain a substantial number of time series, of different natures. The most recent datasets (stemming from the M-4 competition) are worth mentioning, as the resulting methods are publicly available[5] and can be used as benchmarks for the validation and evaluation of the approach proposed in this report. The M4 dataset consists of 100,000 time series coming mainly from Economic, Finance, Demographic and Industry areas, while also including Transportation, Natural Resources, Environment and other domains. The minimum number of observations for each time period is: 13 for yearly; 16 for quarterly; 42 for monthly; 80 for weekly; 93 for daily; 700 for hourly.

[1]`https://opensource.org/licenses/BSD-3-Clause`
[2]`https://www.apache.org/licenses/LICENSE-2.0`
[3]`https://opensource.org/licenses/MIT`
[4]`https://www.gnu.org/licenses/gpl-3.0.html`
[5]`https://github.com/M4Competition/M4-methods`

3.1.2 UCI Machine Learning Repository

The University of California Irvine (UCI) Machine Learning Repository started as an FTP archive in 1987, created by David Aha and graduate students at UCI, and retains a collection of databases, domain theories and data generators that serve as benchmarks for machine learning implementations. For the matter at hand, the repository provides both univariate and multivariate time series datasets which may be used for the validation and evaluation of the approach proposed in this report.

3.2 Tools

The approaches enumerated in chapter 2 aggregate a few of the state of the art options when considering forecasting and anomaly detection practices applied to time series data. The programming frameworks and tools in use to implement these approaches play an important part, as some present richer development environments and simplified deployment. It is important to consider machine learning libraries that feature good runtime performance, tools support, a large community of developers and a healthy ecosystem of supporting packages. This section provides a few details on these tools.

3.2.1 Scikit-learn

Scikit-learn[6] is a Python machine learning library that features algorithms to handle classification, regression, clustering and dimensionality reduction problems. Data pre-processing and model selection modules are also provided. It is designed to interoperate with the Python numerical and scientific libraries NumPy[7] and SciPy[8]. Scikit-learn is licensed under the New Berkeley Software Distribution (BSD) License.

3.2.2 TensorFlow

TensorFlow[9] is an open-source library, created by Google Brain, for numerical computation and large-scale machine learning. It bundles together a large selection of machine learning algorithms, mainly divided by deep learning approaches (Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), etc) and everything else (Support Vector Machine (SVM), k-Nearest Neighbours (k-NN), etc). TensorFlow uses Python to provide a front-end API for building applications with the framework, while executing those applications in high performance C++. TensorFlow is licensed under the Apache License 2.0.

3.2.3 Weka

Waikato Environment for Knowledge Analysis (WEKA)[10] is a Java library containing machine learning algorithms for data mining tasks, while also being able to handle big data problems and perform deep learning. It includes tools for data processing, classification, regression, clustering and data visualization. WEKA is licensed under the GNU General Public License.

[6]`https://scikit-learn.org/stable/`

[7]`http://www.numpy.org/`

[8]`https://www.scipy.org/`

[9]`https://www.tensorflow.org/`

[10]`https://www.cs.waikato.ac.nz/ml/weka/`

3.2.4 PyTorch

PyTorch[11] is a Python based library built to provide flexibility as a deep learning development platform. It has many similarities to TensorFlow, such as hardware-accelerated components and a highly interactive development model that allows for design-as-you-go work. PyTorch is BSD-style licensed.

3.2.5 CNTK

The Microsoft Cognitive Toolkit (CNTK)[12] is a deep learning toolkit that describes neural networks as a series of computational steps via a directed graph. CNTK allows users to easily realize and combine popular model types such as CNNs and recurrent networks (RNNs and Long Short Term Memorys (LSTMs)). It handles many neural network jobs fast, and has a broad set of Application Programming Interfaces (APIs) (Python, C++, C# and Java). CNTK is licensed under the Massachusetts Institute of Technology (MIT) License.

3.2.6 Apache MXNet

Apache MXNet[13] is a deep learning framework that supports state of the art deep learning models (CNNs and LSTMs) and is widely used by Amazon Web Services (AWS). Given its use of a distributed parameter server (based on research at Carnegie Mellon University, Baidu and Google), it can achieve almost linear scale with multiple Graphics Processing Units (GPUs) or Central Processing Units (CPUs). MXNet also supports a broad range of APIs (Python, C++, Scala, R, JavaScript, Julia, Perl and Go). MXNet is licensed under the Apache License 2.0.

Table 3.1 enumerates the previously described tools.

Table 3.1: Tools comparison table describing tools' details.

Tool	**Initial Release**	**Written in**	**Interface**	**Software license**	**Number of GitHub Followers**
Scikit-learn	2007	Python	Python	New BSD	1327
TensorFlow	2015	Python, C++,	Python, C/C++, Java, Go, JavaScript, R, Julia, Swift	Apache 2.0	2043
Weka 3	1997	Java	Java	GNU General Public License	-
PyTorch	2016	Python, C,	Python	BSD	1080
CNTK	2016	C++	Python, C++, BrainScript	MIT license	199
Apache MXNet	2015	C++ core library	C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl	Apache 2.0	706

[11]https://pytorch.org/
[12]https://www.microsoft.com/en-us/cognitive-toolkit/
[13]https://mxnet.apache.org/

Chapter 4

Requirements and Risk Analysis

This chapter presents the proposed approach for development in the context of the internship in regards to its functional and non-functional requirements and associated risks, as well as their analysis.

4.1 Requirements

This section presents the requirements that have been defined for the project. Prioritization for the functional requirements follows the MoSCoW scale:

1: Must have;

2: Should have;

3: Could have;

4: Won't have.

4.1.1 Functional Requirements

The baseline functional requirements collected for the time series analysis framework are present in Table 4.1.

Table 4.1: Baseline functional requirements.

ID	Name	Description	Dependencies	Priority
FR.01	Implement data processing capabilities	The framework should handle data preparation and processing for use in the framework.	-	1
FR.02	Implement a Forecasting framework	Provide a forecasting interface that manages different forecasting approaches for differently defined time series and selects the most appropriate approach, appropriately parameterized.	FR.01	1
FR.03	Implement a Anomaly Detection framework	Integrate an anomaly detection framework that, similarly to the forecasting framework, manages different anomaly detection approaches and outputs the most appropriate, given the input dataset's features.	FR.01	1
FR.04	Implement a REST API to support the framework operations	The implemented framework should be accessible through REST calls in a wrapping API.	FR.01, FR.02, FR.03	2

Implement data processing capabilities The framework should be able to provide and apply data processing techniques to clean and prepare the input dataset for processing within the framework.

Implement a Forecasting framework The framework should provide a forecasting framework that handles both baseline and optimized forecasting approaches, for both single and multi value time series, being able to select the most appropriate approach, based on the features of the given time series.

Implement a Anomaly Detection framework Similarly to the forecasting framework, the anomaly detection framework specializes in anomaly detection approaches.

Implement a REST API to support the framework operations The framework should be accessible through REST API that provides all the operations needed to use the previously mentioned forecasting and anomaly detection frameworks.

Implement data processing capabilities (FR.01)

Some time series analysis methods require the data to incorporate, or not, certain features. As previously discussed in the introduction to chapter 2, time series are usually categorized in relation to seasonality, trend and outliers. Thus, this requirement states the use of techniques that identify and, if necessary, remove components such as seasonality or trend. This requirement also contemplates the handling of other time series features, such as missing values and noise existence. Table 4.2 presents the functional requirements for the implementation of data processing.

Table 4.2: Requirements for Implementation of data processing techniques.

ID	Name	Description	Dependencies	Priority
FR.01.01	Process physical data features	The framework should organize the raw data into a tabular format, recognized by the approaches.	-	2
FR.01.02	Process time series data features	Support for time series specific data features should be provided.	FR.01.01	2
FR.01.03	Transform processed data	Provide techniques to handle standard statistical data transformations, as well as ML transformations.	FR.01.01, FR.01.02	2

Process physical data features It is important for the data to be presented to the framework operations in a correct format. With that in mind, this requirement is responsible for handling inconsistencies related to data entry errors.

Process time series data features As important as it is to guarantee a clean dataset in respect to data errors, this is also valid for data abnormalities in time series structures. This requirement states that features such as missing data and noise are addressed properly before the data is used in a model.

Transform processed data In statistics, data transformations are used to improve graphical readability and, in case of normalization, to rescale data instances into relative values. This requirement states that the framework should be responsible for performing these transformations.

Process time series data features (FR.01.02)

While some of the integrated approaches are more robust, one has to account for the remaining, whose performance is degraded in the presence of phenomenons, such as a high noise level, the presence of outliers, the existence of gaps in the data and non-stationarity. Table 4.3 details what time series data features are relevant to process.

Table 4.3: Requirements for time series data features processing.

ID	Name	Description	Dependencies	Priority
FR.01.02.01	Non-stationarity is addressed	Data processing techniques include measures to detect and remove seasonality and trend.	-	3
FR.01.02.02	Detect and manage missing data	In the presence of missing data instances, interpolation should mask these occurrences.	-	2

Implement a Forecasting framework (FR.02)

In Table 4.4 the functional requirements for the implementation of the forecasting framework are presented.

Table 4.4: Requirements for Implementation of a Forecasting framework.

ID	Name	Description	Dependencies	Priority
FR.02.01	Integrate models	The framework should include specialized approaches whose preference depends on the data.	-	1
FR.02.02	Implement Autonomous Approach Selection	Implement support to automatize the decision of the best forecasting approach, including model parametrization.	FR.02.01	1

Integrate models Baseline methods are independent of the problem context but their quality serves only as a minimum threshold, while more sophisticated approaches can provide better results. This requirement suggests the integration of those methods. For more information on these, one should revisit chapter 2.

Implement Autonomous Approach Selection The forecasting framework should be able to autonomously select the most appropriate approach, as well as select the best parameters to handle the given time series, considering the time series' features.

Integrate models (FR.02.01)

The optimized models can be applied to both Univariate Time Series (UTS) and Multivariate Time Series (MTS). Table 4.5 presents the requirements for optimized models.

Table 4.5: Requirements for models' integration.

ID	Name	Description	Dependencies	Priority
FR.02.01.01	Integrate models for UTS	Implemented optimized models should be appropriated for UTS.	-	2
FR.02.01.02	Integrate models for MTS	Implemented optimized models should be appropriated for MTS.	-	1

Implement Autonomous Approach Selection (FR.02.02)

The previously mentioned approaches should all be available within the framework. However, when contemplating a model choice, one has to account for model fitting. It is expected that, given a small dataset, approaches such as Autoregressive Integrated Moving Average (ARIMA) and Long Short Term Memory (LSTM) would underperform. The autonomous approach selector results as a set of requirements that describe how this component manages the decision of the model, and its parametrization. Table 4.6 details its requirements.

Table 4.6: Requirements for the autonomous approach selector implementation.

ID	Name	Description	Dependencies	Priority
FR.02.02.01	Select the most fitting model	The approach selector should be able to analyze the available approaches in respect to their quality, given the dataset and its features.	-	1
FR.02.02.02	Select the most fitting parameter values	After selecting an approach, the selector should parametrize it in order to maximize the results' quality.	FR.02.03.01	1

Select the most fitting model The functional requirement FR.02.01 provides access to the approaches studied in the state of the art and made available through optimized implementations in the studied toolkits. This requirement then states that the approach selector will choose from a list of approaches, given the datasets' features.

Select the most fitting parameter values Once an approach is selected, it needs to be appropriately parametrized. This is imperative for all of the available methods. In case of k-NN, one should expect few parameters (only the number of neighbours), while other approaches may warrant more parameters, and, therefore, a larger search space for error minimization.

Implement a Anomaly Detection framework (FR.03)

The anomaly detection framework requirements essentially mimic the forecasting framework. The difference lies in the models which the selector opts from.

Implement a REST API to support the framework operations (FR.04)

The time series analysis framework results of an aggregation of the forecasting and anomaly detection frameworks, as well as their data processing and model management capabilities. This wrapping framework is a REST API which provides all of the system's operations and enables easy integration within other services.

4.1.2 Performance Requirements

It is important that the Approach Selector takes the processing time for each algorithm into account, as some algorithms are more complex then others, thus taking more time to execute. The Approach Selector itself should fit in a certain time frame for selecting the most appropriate approach. Table 4.7 presents the proposed performance requirements.

Table 4.7: Performance requirements.

ID	Name	Description	Dependencies	Priority
PR.01	Approach processing time	Processing time for each approach should be taken into account by the Approach Selector.	-	3
PR.02	Selection processing time	The implementation of the Approach Selector should reduce the time for the selection as much as possible.	-	2

Approach processing time This document explores Machine Learning (ML) approaches whose computational complexity ranges greatly, since some approaches have a bigger execution time. This requirement states that the model's implementation should meet a satisfactory performance threshold in order to be usable. In that sense, a ratio relating quality with complexity of the algorithm should be considered, and it should be available to be adjusted by the user.

Selection processing time Each of the selectors have at their disposal a wide selection of algorithms to deal with different problems in time series analysis. The decision for the most appropriate method to use should impact execution time as little as possible. In that sense, this requirement should provide a threshold to consider for the execution time of the autonomous selector.

4.1.3 Technological Requirements

The collected technological requirements stem from chapter 3. In that sense, the following requirements detail both resources, tools and implementation. Table 4.8 presents the technological requirements in place for the project.

Table 4.8: Technological requirements.

ID	Name	Description	Dependencies
TR.01	Programming Language	The project should be developed in Java or Python.	-
TR.02	Open-source Licenses	Licenses from all used software should allow its use in a commercial environment, without additional charge. Allowed licenses are: BSD, Apache, GPL and MIT.	-

4.1.4 Quality Requirements

It was shown in chapter 2 that, for both forecasting and anomaly detection practices, different approaches, with different parametrization, present different results. However, given that some of the gathered datasets already provide satisfactory solutions, it is of interest to attain a relative threshold of quality. Table 4.9 shows the quality requirements at hand for the project.

Table 4.9: Quality requirements.

ID	Name	Description	Dependencies
QR.01	Optimum Threshold	The framework should define a minimum accuracy according to a given benchmark.	-

Optimum Threshold This requirement takes advantage of the already existing solutions, for the datasets in use, to specify a quality threshold given by the metrics they use. This presents an advantage, as the system can be calibrated to produce results with comparable quality to the M4-Competition resulting methods, as well as other implementations provided by developers resorting to the University of California Irvine (UCI) repository.

4.2 Risks

A project is always susceptible to risks that can compromise its development, which is why it is important to reflect on them, and their outcome, and plan mitigation strategies. This section focuses on risk enunciation, as well as their mitigation plans and how they can minimize the consequences of the risks' impact. Figure 4.1 maps the enumerated risks into a risk exposure matrix, in order to provide insight into the risks' probability of happening as well as the impact they carry.

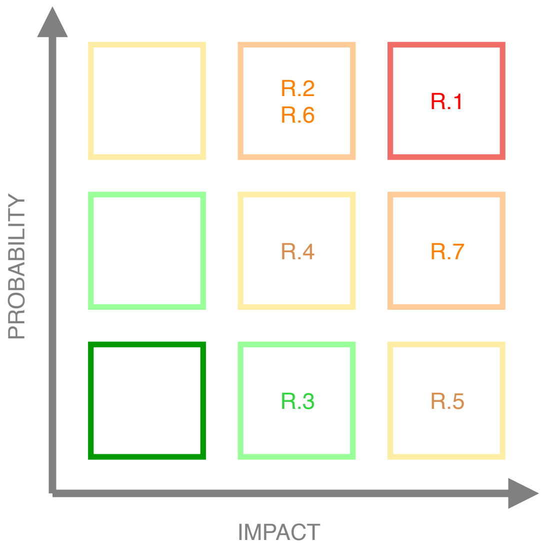


Figure 4.1: Risk exposure matrix.

R.1 - Performance issues in the Autonomous Approach Selector

Both the forecasting and the anomaly detection autonomous approach selector implementation could have a high computation time.

Contingency plan: Reach for a trade-off solution between computational complexity and approach selection quality.

Activation: Indeed, it was necessary to address this issue, in particular, for MTS, where the number of observations and features was much larger than their UTS counterpart, leading to implementation of dimension thresholds to avoid memory errors and decrease the optimization time.

R.2 - Implementation of the Autonomous Approach Selector

The implementation of the Autonomous Approach Selector can be revealed to be more complex than expected, causing a delay on the project development.

Contingency plan: Simplify the requirements for the selector, should there be no workaround.

Activation: As a result of delays during model integration, the Autonomous Approach Selector development suffered. Without a workaround, the conflicting models ended up being discarded from this iteration of the framework.

R.3 - Integrated Approaches' quality

Both forecasting and anomaly detection approaches could deliver sub-par quality in their models.

Contingency plan: Refer to the benchmarks in use, in order to either maintain or optimize the integrated models.

Activation: With the help of expert data science colleagues, the approaches ended up being tested ad-hoc and enhanced with procedures that minimize overfitting and training time.

R.4 - Unavailability of desired approaches' implementation

There is no implementation of a desired approach that we want to use.

Contingency plan: A first approach considers resorting to implementing the approach ourselves. Should this not be possible within the considered time window, the approach might not be considered for the framework.

Activation: There was no reason to activate this contingency plan.

R.5 - Background knowledge

The intern had no previous experience with time series processing nor application of ML techniques on them.

Contingency plan: The work colleagues have experience in the area and may help clarify where to gather information.

Activation: Further investigation regarding various phases of the ML pipeline applied specifically to time series was necessary.

R.6 - Resource Unavailability

The collected repositories include a multitude of datasets, but they may specifically lack time series datasets.

Contingency plan: Search additional dataset sources and/or consider artificial dataset creation.

Activation: An additional search was deemed necessary in order to guarantee a minimum number of examples were fed to the Autonomous Selectors.

R.7 - Computational resources

The framework requires a dataset composed of time series labeled with their optimal solution, which has to be created locally. This process requires each time series to be analyzed by all the models integrating each analysis type. If a large time series is provided, the computational resources increase.

Contingency plan: Allocate local machines in the internship company and/or extend this to Department of Informatics Engineering (DEI) machines.

Activation: It was necessary to allocate one additional machine in order to produce results within the available time window.

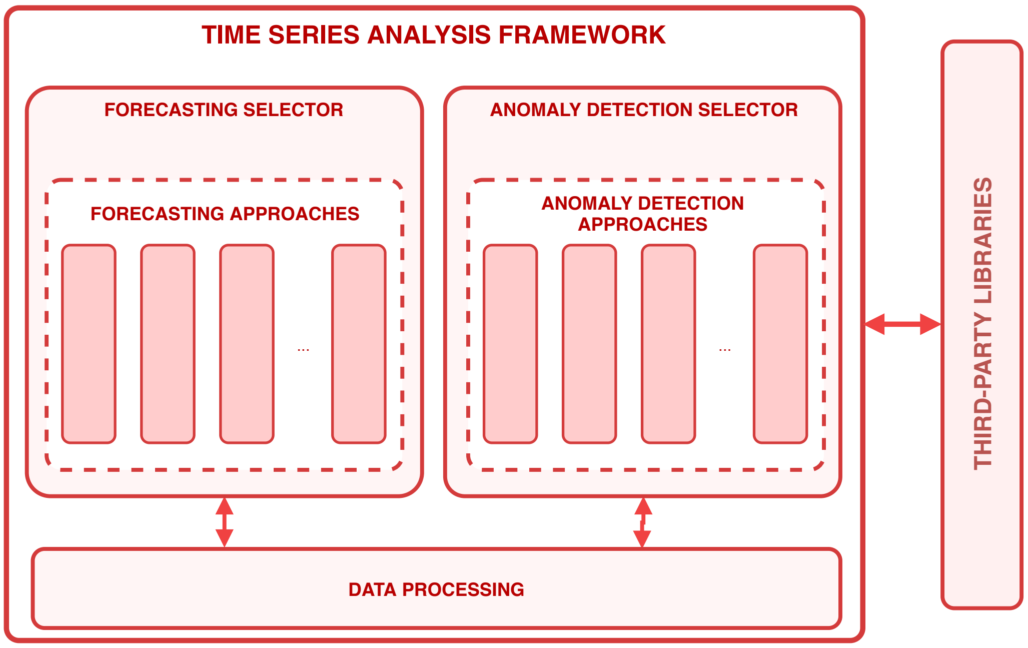
Chapter 5

Approach

This chapter details how the framework's components were orchestrated during the implementation phase of the internship. he emphasis lays on the model selection and model hyperparameter tuning. First, the architecture based on the requirements is projected, followed by a technical walkthrough of Case-based Reasoning (CBR) and Bayesian Optimization applied to the approach.

5.1 Architecture

This section presents the general architecture for the proposed approach. Figure 5.1 shows the modules that compose it.



TIME SERIES ANALYSIS FRAMEWORK
FORECASTING SELECTOR
FORECASTING APPROACHES
...
ANOMALY DETECTION SELECTOR
ANOMALY DETECTION APPROACHES
...
DATA PROCESSING
THIRD-PARTY LIBRARIES

Figure 5.1: General architecture for the framework.

Forecasting Selector

The forecasting selector is responsible for managing operations based on data for which the user requests forecasting. It maintains a set of forecasting approaches that can be chosen and properly parameterized (specified in the model management component shown in the architecture in figure

5.1) to meet the data features. Model selection is ensured by the implemented CBR mechanisms, by comparing the current case with previous cases in the case-base.

The selector contains optimized approaches, which are detailed in chapter 2, that can take advantage of the problem's context. The model selection is divided into UTS and MTS scenarios. In the UTS scenario, model selection is ensured by a CBR k-Nearest Neighbours (k-NN), whereas the MTS model selection is provided by CBR based on Semi Metric Ensemble Time Series (SMETS), detailed in Section 5.3.2. Redirecting to CBR's types of reasoning, the proposed approach is classified as instance-based reasoning. The Case Base associated with this approach follows a clustered organization. The univariate selector itself is represented through a k-NN, while the multivariate counterparts are expressed through a novelty metric, further detailed in section 5.3.2. In this scenario, a problem case is represented through a dataset's time series features, such as the total number of instances, number of missing instances and others, while the case solution holds information on the best performing algorithm, its hyperparametrization, its confusion matrix, and its total execution time.

Anomaly Detection Selector

The anomaly detection selector is similar to the forecasting selector, but it handles and provides anomaly detection approaches, applied on MTS.

Data Processing

The data processing module applies techniques in order for the data to be correctly ingested by the analysis selectors and to enhance the performance and accuracy of the approaches. The techniques which integrate this component also stem mainly from the provided third-party libraries, and include operations that handle data features:

`Pandas`[1]: Used for data ingestion in a tabular format manipulable by common Python libraries such as NumPy [2] and for data imputation, to remove inconsistencies not handled within the provided selection of approaches.

`scikit-learn`: Used for dataset feature reduction, in order to maximize performance while minimizing the loss of information and for dataset feature ranking, to focus on the features with the most discriminative power, while addressing the Forecasting and Anomaly Detection Selectors.

5.2 Overview

The product developed during the internship comprises a framework that receives time series as input, and, given a type of analysis (forecasting or anomaly detection) returns the best performing model for that sequence of data. As illustrated in figure 5.2, the framework is prepared to handle both UTS and MTS when the user wants to perform forecasting, and MTS when the preferred analysis is anomaly detection. Details on the datasets in use to build the case-base are present in section 5.2.1.

[1]`https://pandas.pydata.org/`
[2]`https://www.numpy.org/`

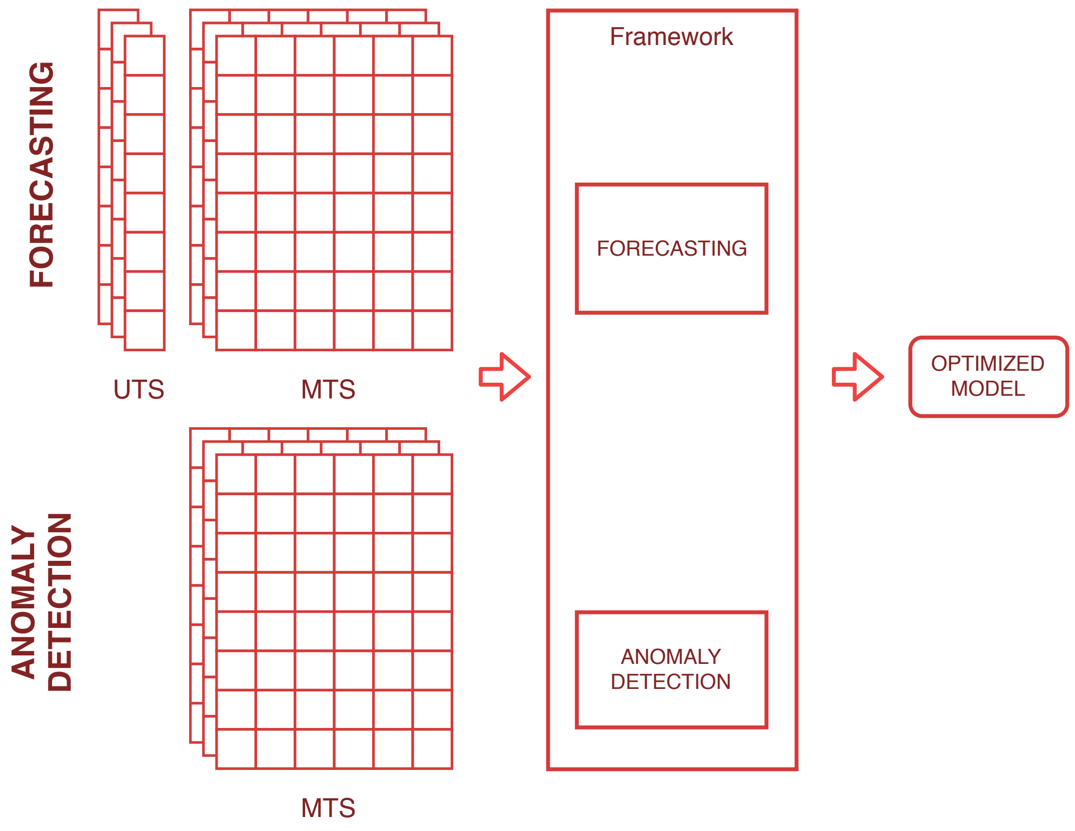


Figure 5.2: An overview of the input and output of the framework.

The framework itself can be further explained through the CBR cycle, described in Section 2.3 and its integrating phases, specified in Figure 5.3. The retrieve phase is responsible for fetching the most similar example from the case-base, but this process is different between time series types (UTS and MTS). For this reason, a case-base for each combination of analysis and time series types is necessary. For example, there is a case-base for UTS used for forecasting. Similar UTS cases are processed through a k-NN, which considers the closest neighbor in its prediction. In order to train the k-NN, the framework integrates an external, open-source library known as Time Series Feature extraction based on Scalable Hypothesis Tests (TSFRESH)[3] responsible for feature extraction, which is described in section 5.3.1. Regarding the case of MTS, it is not possible to use k-NN due to the different number of features between MTS. Therefore, a semi-metric, SMETS, proposed in [85] is used.

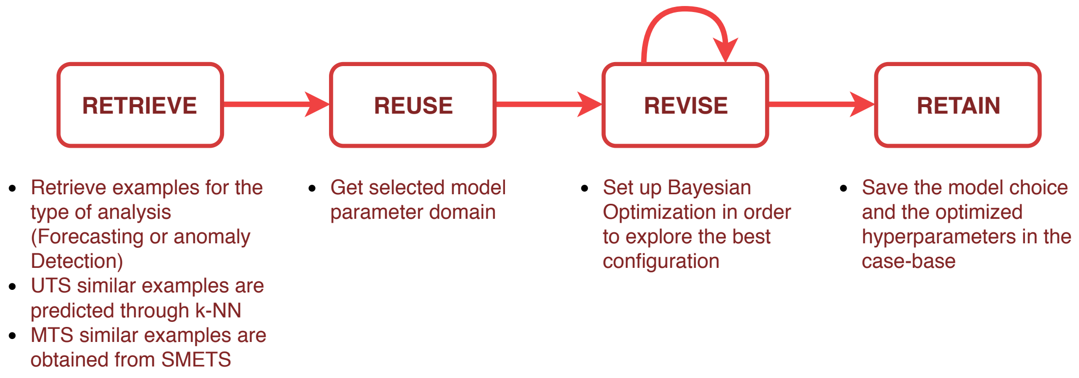

RETRIEVE
REUSE
REVISE
RETAIN
• Retrieve examples for the type of analysis (Forecasting or anomaly Detection)
• UTS similar examples are predicted through k-NN
• MTS similar examples are obtained from SMETS
• Get selected model parameter domain
• Set up Bayesian Optimization in order to explore the best configuration
• Save the model choice and the optimized hyperparameters in the case-base

Figure 5.3: An overview of the input and output of the framework.

[3] `https://tsfresh.readthedocs.io/en/latest/index.html`

5.2.1 Datasets

The CBR case-bases are built with UTS for forecasting and MTS for forecasting and anomaly detection. In this section, the list of used time series is provided, as well as details regarding both the datasets and the pre-processing involved.

Aside from specific dataset modifications (dimensionality reduction), all datasets in this study were formatted to a specific format of input to the framework.

1. All datasets are read into a `Pandas DataFrame` structure. This is essentially a matrix structure;
2. All datasets were checked for errors (`NaN` and invalid column type) and then imputed. The performed imputation replaced missing data with the last valid value in the column;
3. The datasets' index is replaced by a `DateTimeIndex` object which acts as a timestamp. This is especially useful to take advantage of the `Pandas` time series methods and to perform operations on the frequencies, such as uniformize the sample rate and resample.

Forecasting

As discussed in chapter 3, the Makridakis Competitions (M-Competitions) play an important part in forecasting research. These competitions have an extensive repository of datasets to experiment on, but all of them are UTS. Since they don't provide any MTS datasets, the UCI Repository was responsible for providing this type of dataset.

Forecasting UTS The UTS in use for forecasting were made available by one of the M-Competition iterations, namely the third. The dataset made public[4] is composed of a total of 3003 time series mainly separated by frequency, through spreadsheet tabs. The frequency ranges from yearly to monthly, while the last 173 do not have a defined frequency. During this project, these time series are assigned a daily frequency. Each tab provides information on the time series ID, its length, the category and the starting date/time. The time series' exact type is shown in Table 5.1.

Table 5.1: M3-competition data type distribution showing the types of time series data per interval.

	Types of Time Series Data						
Interval	**Micro**	**Industry**	**Macro**	**Finance**	**Demog**	**Other**	**Total**
Yearly	146	102	83	58	245	11	645
Quarterly	204	83	336	76	57	0	756
Monthly	474	334	312	145	111	52	1428
Other	4	0	0	29	0	141	174
Total	828	519	731	308	413	204	3003

Forecasting MTS Multivariate datasets for forecasting were retrieved from the UCI repository. Table 5.2 provides information regarding the datasets in use.

[4]`https://forecasters.org/resources/time-series-data/m3-competition/`

Table 5.2: Name, Dimensionality (observations × features) and Frequency of the multivariate forecasting datasets in use.

Name	Dimensionality	Frequency
Air Quality[5]	9358 × 15	Hour
Appliances Energy[6]	19735 × 29	Minute
Beijing PM 2.5[7]	43824 × 13	Hour
Hydraulics[8]	2205 × 43680	Minute
Flights[9]	[4664, 4896] × 30	Second
User Stats[10]	169 × 4	Hour

The goal for the experiments carried out during the internship was to use datasets with varying dimensionality and frequency. Specific pre-processing measures being addressed per dataset are presented below:

- Starting with the Air Quality dataset, the last 5 columns were removed, as there is no information regarding their purpose. This dataset has missing values, which warrants interpolation from the last valid value recorded, for each feature;
- The Flights datasets are actually 10 separate MTS sampled in the same context, but they differ in the number of observations. The smallest flight has 4664 observations, while the largest has 4896;
- The Hydraulics dataset is another case where the data was internally manipulated in order to obtain a dataset that registered a dynamic behavior (non-constant observation values). While the original length of the dataset is 2205, the dataset observations were reduced to 36. Next, there are several readings for each component of this dataset, accounting to a total of 43680 features. However, this study relied solely on the profile of the main component, which is composed of only 5 features. The final dimension of this dataset is, then, 36 × 5.

Besides these specific cases, all datasets were processed in order to extract (or generate) a `DateTimeIndex`.

Anomaly Detection

For Anomaly Detection, the features are used to determine if an instance is an outlier or not. In the UCI repository, some datasets of the MTS type following this structure were found. Table 5.3 highlights the datasets used for anomaly detection.

[5] `https://archive.ics.uci.edu/ml/datasets/Air+quality`
[6] `https://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction`
[7] `https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data`
[8] `https://archive.ics.uci.edu/ml/datasets/Condition+monitoring+of+hydraulic+systems`
[9] `https://c3.nasa.gov/dashlink/resources/449/`
[10] `https://www.kaggle.com/wolfgangb33r/usercount/version/4`

Table 5.3: Name, Dimensionality (observations × features) and Frequency of the multivariate anomaly detection datasets in use.

Name	Dimensionality	Frequency
Daphnet[11]	[38774, 195737] × 10	Millisecond
Eye State[12]	14980 × 15	Second
Health[13]	98304 × 24	Millisecond
Occupancy[14]	8143 × 6	Minute
Ozone[15]	2536 × 73	Day

As was the case for the forecasting datasets, anomaly detection datasets were prone to slight modifications in order to be compatible with the framework structure. The Daphnet datasets were built from patient readings, using several key locations in the human physiology. There are a total of 10 patients, and, thus, 10 datasets to account for. This dataset collection is another example where the number of observations differs between patient datasets. In order to benefit from anomaly detection in these datasets, their "*Annotation*" label was merged to a binary format: Non-anomalies are represented as 0; Anomalies are represented as 1. This transformation was also applied to the Health dataset.

UTS for anomaly detection represent problems that are not frequent in the literature [66] which severely impacts CBR retrieval results.

5.2.2 Metrics

CBR decision leads to an input time series being assigned an optimal model, given an analysis type (forecasting or anomaly detection). This model is then fitted and tuned to optimize the output of its intended analysis. In all cases, the generated outputs need to be evaluated in relation to what is actually true. In this section, the metrics used for the CBR model classification, the forecasting regression and the anomaly detection classifications are presented and detailed.

CBR

Classification on the CBR algorithm (k-NN for UTS and SMETS for MTS) is multiclass, covering up to 7 possible labels (model solutions) per analysis type. Table 5.4 details the metrics being collected for this classification problem.

Table 5.4: CBR metrics' name, description and application formula (using True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN)) in use.

Metric	Description	Formula
Accuracy	Proportion of true labels over all labels.	$\frac{TP+TN}{TP+TN+FP+FN}$
Recall	Proportion of true positives over all actual positives.	$\frac{TP}{TP+FN}$
Precision	Proportion of true positives over all predicted positives.	$\frac{TP}{TP+FP}$
F1-score	Harmonic average of the Precision and Recall.	$2\frac{Pr\times Re}{Pr+Re}$

[12]https://archive.ics.uci.edu/ml/datasets/Daphnet+Freezing+of+Gait
[13]http://archive.ics.uci.edu/ml/datasets/mhealth+dataset
[14]https://archive.ics.uci.edu/ml/datasets/Occupancy+Detection+
[15]https://archive.ics.uci.edu/ml/datasets/ozone+level+detection

The Accuracy is an easy metric to understand, but it is also misleading, because it does not account for imbalanced datasets [73]. This can be very problematic in scenarios where the cost of misclassifying an actual positive, or an actual negative, is very high. This is where Recall and Precision emerge. Precision is a good option when the cost of false positives is high [73]. For example, there is a model that classifies an e-mail as spam. If the e-mail is incorrectly classified as spam, the e-mail user may lose important information. In the case of Recall, this metric focuses on evaluating a model where the false negatives have a high cost [73]. Take as an example a sick patient scenario: There are patients that developed a sickness, but the model classified one of these patients as not sick. The cost of the false negative will be very high if the sickness is contagious. In order to account for both cases presented by the previous metrics, F1-Score is presented. This metric's goal is to establish a balance between Precision and Recall all the while contemplating class imbalance [73]. Joining these metrics results in a lower weight on the negative class, which is the one with lowest importance.

The previously mentioned metrics employed on the CBR algorithm evaluation deal with multiclass targets. In this particular case, it is important to define the average computation. Table 5.5 shows the methods provided by `scikit-learn` used to obtain the average between classes.

Table 5.5: Averaging methods in use.

Average	**Description**
`micro`	Calculates metrics globally by counting the total TP, FN and FP.
`macro`	Calculates metrics for each class and outputs their unweighted mean.
`weighted`	Calculates metrics for each class and outputs their mean weighted by the number of true instances for each class.

The `micro` average is preferable when there appears to be class imbalance, whereas `macro` average does not take imbalance into account and rather focuses on global quality. The `weighted` average results as an adaptation of `macro`, where each classes' metric is weighted by the relative number of examples available for it. All of the mentioned averaging types are calculated for the F1-score, Precision and Recall. Note that in a multiclass scenario, using the `micro` average, Precision and Recall will output the same values due to both metrics counting all false instances.

Forecasting

Forecasting predictions carried out within this framework are numerical, i. e., their distance to the actual true value has meaning. Therefore, forecasting metrics are based on errors. Table 5.6 details the errors being calculated within forecasting model evaluation.

Table 5.6: Forecasting metrics' name, description, application formula and scale dependency in use.

Metric	**Description**	**Formula**	**Scale-dependent**
MAE	Average absolute difference between $\hat{y}$ and y.	$\frac{1}{n}\sum_t^n \lvert\hat{y}_t - y_t\rvert$	Yes
MSE	Average squared difference between $\hat{y}$ and y.	$\frac{1}{n}\sum_t^n (y - \hat{y}_t)^2$	Yes
sMAPE	Average absolute difference between $\hat{y}$ and y scaled by the sum of the absolute y and $\hat{y}$ in percentage.	$\frac{100\%}{n}\sum_t^n \frac{\lvert F_t - A_t\rvert}{\lvert A_t\rvert + \lvert F_t\rvert}$	No
rMSE	Square root of the MSE.	$\sqrt{MSE}$	Yes

Mean Squared Error (MSE) is useful when there are observation spikes that require more attention [42]. However, if one single bad prediction is made, this metric heavily overestimates a bad model, due to the squaring factor. This is frequent in noisy data, where, if a perfect model exists, it returns a high MSE. The symmetric situation is also problematic. If all errors are small, the model badness may be underestimated.

From MSE, it is possible to obtain **Root Mean Square Error (RMSE)**, where the square root is introduced to apply the scale of the targets to the scale of the errors.

Mean Absolute Error (MAE) describes an average absolute difference which means that all the individual differences are weighted equally [42]. For example, the difference between 20 and 0 is twice the difference between 10 and 0. Using MSE, the metric would believe the error is four times worse, instead of twice. In that sense, MAE is more robust to outliers. In the end, the choice between these metrics relies on the nature of the problem regarding outlier importance: if there are known outliers, there is no need to account for them, but if unknown outliers are present, it is of interest to highlight them, as shown in [43]. Regarding their interpretability, MAE is often easier to understand: it calculates the average absolute error to expect for each prediction. Even so, it is not ideal when comparing different datasets. An MAE of 10 between time series of different value range could mean different model quality.

Experiments that ran on this framework benefit from direct comparison between different time series, which is why **Symmetric Mean Absolute Percentage Error (sMAPE)** is present. sMAPE is an easier metric to interpret because it is scale independent and can be presented as a percentage with values between 0 and 100 [42]. Still, in [35], sMAPE is criticized for its asymmetry, since under-forecasts are penalized heavier than over-forecasts. This reason is the cause for research for a more appropriate metric that shares the benefits of sMAPE, as shown in [25].

Anomaly Detection

As is the case with CBR algorithms, anomaly detection predictions are based on a categorical variable: whether an observation reflects an anomaly, or not. However, this analysis type differs from the former by only accounting for binary classifications. In anomaly detection, all instances (TP, TN, FP and FN) are locally stored, along with the `weighted` F1-Score, due to its mostly stable proportion of precision and recall, as discussed in Subsection 5.2.2.

5.3 Retrieval Phase

The Retrieval Phase of the CBR is divided into 3 different components: The UTS CBR currently available for forecasting, the MTS CBR available for forecasting and the MTS CBR available for anomaly detection. This section focuses on detailing each one of these.

5.3.1 UTS CBR

The UTS CBR component builds (or loads, if locally stored) a k-NN fitted on the case-base. The case-base comprises a dataset, where each observation is a UTS, the features, which are obtained through the use of an external library, TSFRESH, and, finally, the optimal model solution, as well as its metric readings and optimal parameter values, as shown in figure 5.4.

NAME	ABS_ENERGY	...	MEAN	STATIONARY	SOLUTION	SOL_METRICS	SOL_PARAMS

Figure 5.4: UTS CBR dataset header row, showing the index (name), some of the calculated features and the solution columns.

As the name refers, TSFRESH extracts features from time series based on hypothesis tests, such as the total number of peaks, maximum and minimum values, mean and mode. While the list of features is rather extensive. Appendix A includes Table A.1, containing the feature calculators present within the library.

Before effectively entering the retrieve phase within the framework, both the new case and the case-base undergo feature extraction. As stated in the library documentation, most of the generated features may not produce valid values, given their dependency on the length of the time series being studied. Thus, they undergo imputation before being effectively assigned to the time series. Once this step is complete, the following transformations are applied to the data:

Feature reduction: Feature reduction is employed through the use of Pearson Correlation[16], with a threshold held at 90%. Preliminary results showed that with this threshold, the case-base feature number was sufficiently reduced to avoid performance hindrances while taking advantage of the remaining features and their discriminative power.

Feature selection: Feature selection is provided through a `scikit-learn` method, which takes a model of their library with a specific attribute, responsible for attributing weights to each feature used in the fitting phase. For the UTS CBR k-NN, a Linear Support Vector Classifier, with $C = 1.0$, using the l1 penalty and threshold equal to the importance mean was used.

At this point, we have all of the recorded observations as well as all the features which are fed to the model. The formatted data is split into 70/30 splits of training and testing subsets, respectively. To ensure that all of the employed models get visibility, the training data is balanced. Class balancing is assured through the following procedure:

```
mean_observations = len(X_train) // len(labels)
for class in labels:
  class_observations = X_train[X_train['label'] == class]
  if class_observations > mean_observations:
    excess = class_observations - mean_observations
    class_observations.shuffle()
    class_observations.remove(0:excess)
  else:
    random_oversample(class_observations)
```

From here, the model is fitted with folds originated from 10-fold cross validation, in order to obtain the model whose test score is highest. Finally, the model is fitted, and tested with the test split, following the pipeline shown in figure 5.5.

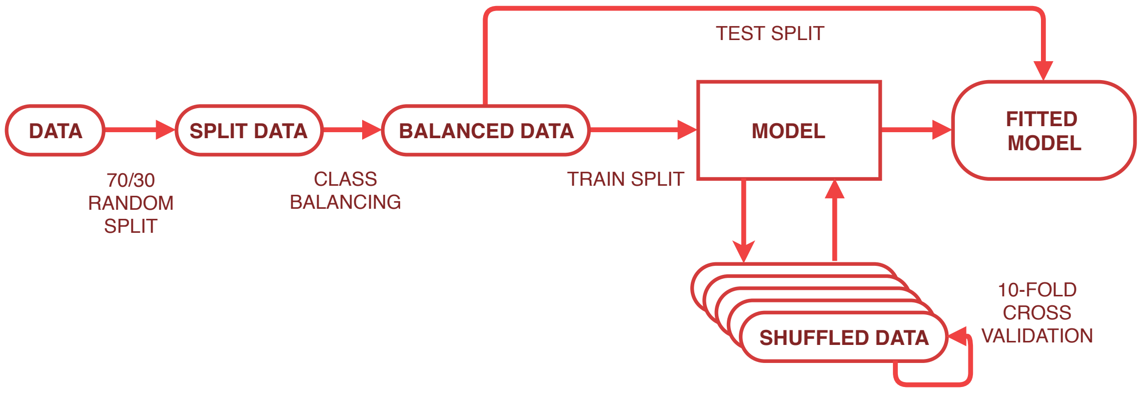


Figure 5.5: Pipeline which leads to the conception of the UTS CBR k-NN.

5.3.2 MTS CBR

It is impossible to attain similarities between different MTSs using k-NN, given that they can have a different number of dimensions (features, that is). However, a semi-metric that calculates

[16]Pearson Correlation calculates the linear correlation between two variable samples, expressed between -1 (negative correlation) and 1 (positive correlation).

similarities between MTSs has recently been proposed [85]. SMETS infers similarity between a pair of MTSs by:

1. Partial matching, where each UTS of the MTSs is pair-wise compared, by the distance to each data point. This phase stops when all UTS from the smaller MTS is assigned to one UTS from the bigger MTS.
2. Penalization for each unassigned UTS from the bigger MTS, by weighting the distance between the unmatched components to the closest counterpart in the other MTS by the proportion of information contained in that counterpart.

The MTS CBR dataset assumes the format shown in Figure 5.6.

NAME	SOLUTION	SOL_METRICS	SOL_PARAMS

Figure 5.6: MTS CBR dataset header row, showing the index (name) and the solution columns.

Integration of SMETS[17] was possible with `oct2py`[18]. In order to maximize performance, the number of observations is limited by 1000, a value decided on a compromise between the smallest and largest datasets collected. While SMETS performs with varying number of features between each dataset, it is not appropriate for MTS with a varying number of observations. The work produced during this internship includes an adaptation of SMETS which handles the differing row dimensions. In order to circumvent this issue, the following approach was taken:

1. It is impossible to account for the sampling rate from any time series, so, to boot, each compared MTS has their frequency checked. The following pseudo-code presents the approach.

   ```
   for i in time_series:
     frequency = time_series(i+1) - time_series(i)
     dataframe_frequency.append(frequency)

     if frequency == previous_frequency:
       count += 1
     if count == threshold:
       break
     else:
       last = freq
       count = 1
   frequency = mode(dataframe_frequency)
   return frequency
   ```

   Do note that the threshold used for early stopping assumes a total of 15 samples, given the maximum considered length of any MTS. Next, each MTS is resampled to the recognized frequency, in order to maintain the frequency consistent for further calculations.

2. Now that we have both MTS correctly sampled, we need to certify that we are able to resample them in order to match the number of observations. Thus, we start by checking if they are in the same frequency unit. If one of the series' is sampled yearly and the other is sampled daily, resampling is necessary. The goal is to work with the same frequency unit. Thus, we first check if it is feasible to decrease the yearly MTS (the one with the highest frequency) to daily samples (the MTS with the lowest frequency). The following pseudo-code provides insight into this process. This serves merely as an example, as the considered frequency units go all the way down to microseconds.

[17] `http://www.comp-sys-bio.org/software/SMETS-1.m`

[18] `https://pypi.org/project/oct2py/`

```
start_unit = 1
while X_frequency > Y_frequency:
  if X_frequency is yearly:
    X_frequency = monthly
    start_unit *= 12
  elif X_frequency is monthly:
    X_frequency = daily
    start_unit *= 30
  elif X_frequency is daily:
    X_frequency = hourly
    start_unit *= 24
  elif X_frequency is hourly:
    X_frequency = minute
    start_unit *= 60
return start_unit < threshold
```

At this point, both MTS are sampled in the same frequency unit.

3. Finally, we want to make sure that both MTS have the same number of observations. The following pseudo-code presents how we adjust the dataset with more observations.

```
proportion = n_obs_bigger_MTS / n_obs_smaller_MTS
bigger_MTS = resample(bigger_MTS, proportion)
```

Given that the calculation of the proportion usually outputs a float, even after resampling the bigger MTS, the number of observations may not match. At this point, we discard the last elements of the bigger MTS until they match.

Once this process is finished, the MTSs can be submitted to predictions by SMETS. Unlike as UTS CBR, this approach is not based on a traditional ML model, therefore, not needing processes such as data balancing. Cross-validation has also been removed due to limitations in integrating SMETS into the cross-validation process provided in `scikit-learn`. The pipeline for MTS CBR is presented in Figure 5.7.

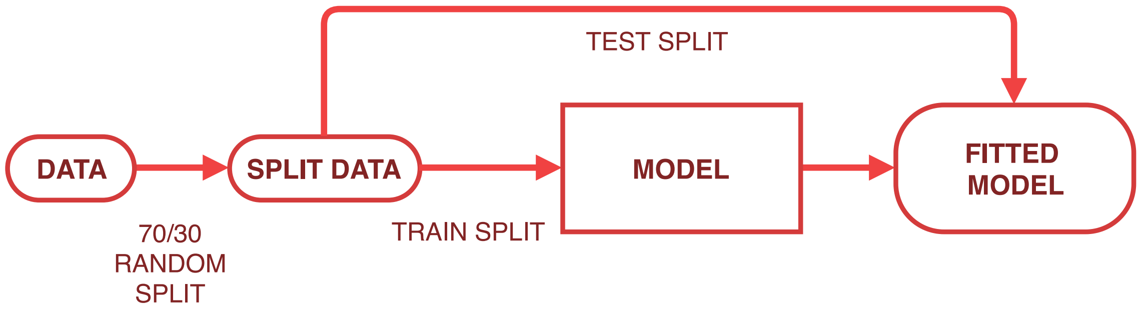


Figure 5.7: Pipeline which leads to the conception of the MTS CBR classification based on SMETS.

5.4 Reuse Phase

The Retrieval Phase of the CBR cycle outputs the most similar problem case and, more importantly, its solution. This phase is followed by the Reuse Phase, where the model configuration is first addressed.

5.4.1 Implemented models

Before delving into the model configuration (and its hyperparameter domain), Tables 5.7 and 5.15 show the integrated models, as well as their original source and the parameters which are explored later during the optimization phase.

Table 5.7: Forecasting models, their source library, and hyperparameters in study.

Model	Source	Hyperparameters in use
MLP Regressor	`scikit-learn`	`n_hidden_layers, n_hidden_units, max_iter, alpha`
LSTM	`keras`	`n_lstm_layers, n_lstm_units`
SVR	`scikit-learn`	`C, epsilon`
k-NN Regressor	`scikit-learn`	`k`
CART	`scikit-learn`	`criterion`
GRNN	`neupy`	`std`
GP	`scikit-learn`	`kernel, alpha`

Most of the aforementioned methods stem from `scikit-learn`. Support Vector Regression (SVR) and Generalized Regression Neural Network (GRNN) warranted a multi-output regressor, provided by `scikit-learn` once MTS were considered. Multi Layer Perceptron (MLP) and k-NN were used in the format of regression problems. All models were also configured with `n_lags` which represents the number of past lags used for the sliding windows during data pre-processing. LSTMs have a particular type of `n_lags`, which we define as `look_back`, which, ultimately, has the same effect. Instead of creating past lag columns, the data is formatted such that each example fed to the network comprises a unidimensional array of the past `look_back` observations, for each variable in the initial time series. In order to control the search space for this type of variable, we calculated a maximum threshold which depends on the original length of the time series. If the series surpasses 100 observations, this threshold maxes out at 15. If the series has less than 100 examples, the threshold is given by half of the smallest split, between the validation and test splits. The search space for `n_lags` ranges from 1 to `smallest_split_length`. Tables 5.8 to 5.14 describe the hyperparameters being tuned during the optimization phase for forecasting models, by bayesian optimization.

Table 5.8: MLP hyperparameters' name, description and value interval being studied.

Hyperparameter	Description	Search space
`n_hidden_layers`	The number of hidden layers to include in the MLP architecture.	[1, 10]
`n_hidden_units`	The number of neurons to append to each hidden layer.	[1, 20 * n_features]
`max_iter`	The maximum number of iterations allowed during fitting.	[100, 3000]
`alpha`	The regularization (penalty) term that avoids overfitting by constraining weight size.	$[1e^{-5}, 1e^{-2}]$

Regarding the number of hidden layers and the number units per layer, a rule of thumb approach was taken. Values between 1 and 10, for the number of layers, should be enough, given that, for UTS, the maximum number of lags would never surpass 15, i.e, we would never process more than 15 'features' per time series. The number of hidden neurons resulted from ad-hoc tests that started from ranges maxing out at 20 neurons. Eventually, results showed that this architecture would behave better for a larger number of neurons, all the while avoiding overfitting. The maximum number of iterations were studied in order to infer if there were cases where the model would behave best given a shorter execution time. However, a maximum threshold of 3000 iterations was considered in order to give the model time to effectively learn. The remaining default parameters for MLP employ early stop maneuvers, should the network enter overfit. The alpha term has a default value of $10e^{-3}$, therefore, to guarantee a wide search space, while avoiding big deviations, the search space maxes out at $10e^{-1}$.

Table 5.9: LSTM hyperparameters' name, description and value interval being studied.

Hyperparameter	Description	Search space
`n_lstm_layers`	The number of LSTM layers to include in the architecture.	[1, 10]
`n_lstm_units`	The number of neurons to append to each LSTM layer.	[1, 20 * n_features]

Similarly to MLP, the architectures defined in the LSTM contain hidden layers, with a variable number of hidden neurons. This architecture's search space is equal to the MLP's. Where it differs from MLP is in how it is architecture is built. `Scikit-learn`'s implementation of MLP builds hidden layers with the same number of neurons. Keras implementations of Neural Network (NN) can include different natured layers (such as regular dense connected layers) or LSTM layers, which are the ones being used. Given the time constraint to explore architectures, the framework focuses on building stacked LSTM architectures. Figure 5.8 presents the final architecture generated for the following configuration:

- `n_lstm_layers` = 5
- `n_lstm_units` = 10
- `n_features` = 3

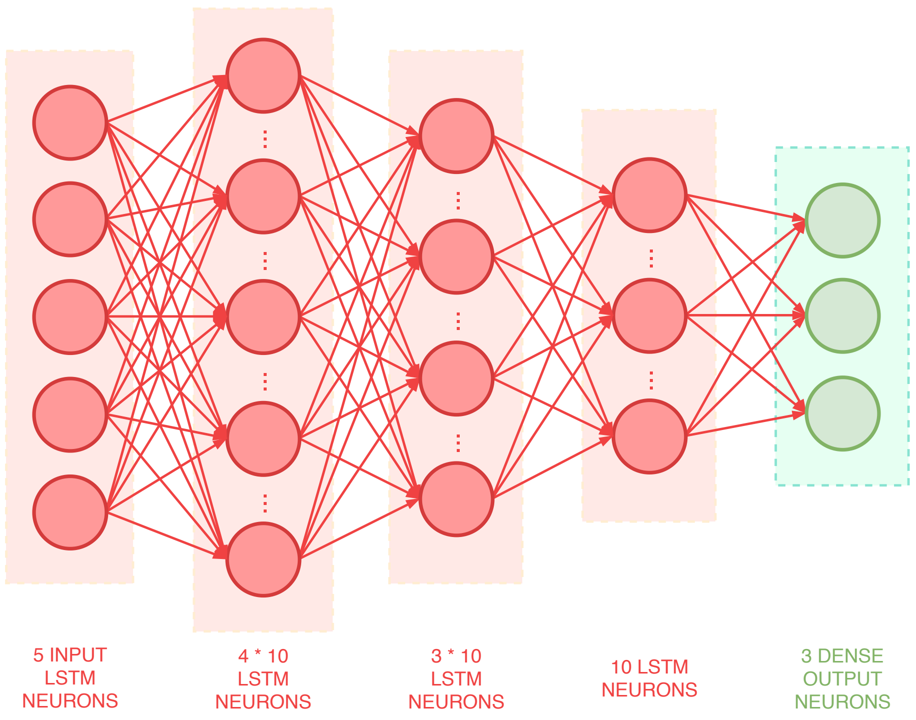


Figure 5.8: Example of an LSTM architecture generated by the framework.

Table 5.10: SVR hyperparameters' name, description and value interval being studied.

Hyperparameter	Description	Search space
`C`	Penalty parameter C of the error term.	[0.01, 100]
`epsilon`	The epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value.	[0.1, 1]

The `C` parameter is responsible for defining the capacity of the model. That is, if manipulated into extremes, it contributes to either a model which correctly classifies each training instance (a complex model, to deal with data which does not vary too much) or a model whose hyperplane is essentially a straight line (leads to many misclassifications). The `C` value is usually situated between $1e^{-6}$ and $1e^{3}$. Given the variation associated to `C`, the maximum `epsilon` is 1% of the highest `C`.

Table 5.11: k-NN hyperparameters' name, description and value interval being studied.

Hyperparameter	**Description**	**Search space**
`k`	Number of neighbors to use by default for `kneighbors` queries.	[1, smallest_split_length]
`weights`	Weight function used during prediction. With 'distance', closer neighbors of a query point will have a greater influence than neighbors which are further away. With 'uniform', all points in each neighborhood are weighted equally.	['uniform', 'distance']

Parameter `k` defines the number of neighbors to include in the neighborhood. The maximum number of neighbors to considered is in direct relation to the smallest split length (which is maxed out as 15). In any case, ad-hoc tests for some of the considered UTS showed that the optimal k never rose beyond 10.

Table 5.12: CART hyperparameters' name, description and value interval being studied.

Hyperparameter	**Description**	**Search space**
`criterion`	The function to measure the quality of a split.	['mse', 'friedman_mse', 'mae']

The criterion is a categorical variable which assumes, at most, 3 different values, according to the latest version of `scikit-learn`. In regards to computational complexity, either choice is similar, and, since no other hyperparameter is currently being considered, we exhaust the criterion hyperparameter.

Table 5.13: GRNN hyperparameters' name, description and value interval being studied.

Hyperparameter	**Description**	**Search space**
`std`	Standard deviation for PDF function.	[0.1, 0.5]

In the library it is introduced in, the GRNN receive exactly one hyperparameter, namely the stand deviation for the PDF function. The authors give an example where the standard deviation should be between 10 and 15 if the input features range from 0 to 20. In this framework, all datasets are scaled to a $[0, 1]$ range. Thus, the std assumes values in the interval $[0.1, 0.5]$.

Table 5.14: GP hyperparameters' name, description and value interval being studied.

Hyperparameter	**Description**	**Search space**
`kernel`	The kernel specifying the covariance function of the GP.	['DotProduct', 'ConstantKernel', 'Matern', 'RBF', 'RationalQuadratic']
`alpha`	Value added to the diagonal of the kernel matrix during fitting. Larger values correspond to increased noise level in the observations.	[0.0001, 0.01]

All kernels used by Gaussian Process (GP) are worth considering, as this algorithm only tests 2 different hyperparameters and this search is not too costly (unlike MLP with its 4 hyperparameters being evaluated). The alpha search space guarantees that we allow a wide range of time series to be as accurately analysed as possible (the framework accepts any time series, in the proper tabular format).

Table 5.15: Anomaly Detection models, their source library, and hyperparameters in study.

Model	Source	Hyperparameters in use
L-SVM	`scikit-learn`	`C, max_iter`
LSTM	`keras`	`n_lstm_layers, lstm_units`
BGMM	`scikit-learn`	`covariance_type`
GMM	`scikit-learn`	`covariance_type`
k-NN	`scikit-learn`	`k, weights`
MLP	`scikit-learn`	`n_hidden_layers, n_hidden_units, max_iter, alpha`
SVM	`scikit-learn`	`C, kernel, max_iter`

Anomaly Detection models are more straightforward, in the sense that they do not benefit from sliding windows, assuming a more traditional data pre-processing, consisting sequential data splits. Tables 5.16 to 5.21 describe the hyperparameters being tuned during the optimization phase for anomaly detection models.

Table 5.16: Linear SVM hyperparameters' name, description and value interval being studied.

Hyperparameter	Description	Search space
`C`	Penalty parameter C of the error term.	[0.0001, 100]
`max_iter`	The maximum number of iterations to be run.	[100, 2000]

The Linear Support Vector Machine (SVM) gets its name from the linear kernel available for the SVM. Other than that, the `C` parameter mimics its forecasting counterpart. The default maximum number of iterations is 1000, which may not be enough. Therefore, it is extended to 2000, but starting from 100.

Table 5.17: LSTM hyperparameters' name, description and value interval being studied.

Hyperparameter	Description	Search space
`n_lstm_layers`	The number of LSTM layers to include in the architecture.	[1, 10]
`n_lstm_units`	The number of neurons to append to each LSTM layer.	[1, 20 * n_features]

As was the case for forecasting, the LSTM architecture configuration remains essentially the same, considering the same hyperparameters.

Table 5.18: Bayesian Gaussian Mixture Model (BGMM) and Gaussian Mixture Model (GMM) hyperparameters' name, description and value interval being studied.

Hyperparameter	Description	Search space
`covariance_type`	The type of covariance parameters to use. With 'full', each component has its own general covariance matrix. In 'tied', all components share the same general covariance matrix. In 'diag', each component has its own diagonal covariance matrix. For 'spherical', each component has its own single variance.	['full', 'spherical', 'tied', 'diag']

The covariance type is the only tested categorical parameter. Initial tests showed that the implementation of these algorithms had complications when modifying the number of components, resulting in exceptions. Therefore, the default value was used.

Table 5.19: k-NN hyperparameters' name, description and value interval being studied.

Hyperparameter	Description	Search space
`k`	Number of neighbors to use by default for kneighbors queries.	[1, smallest_split_length]
`weights`	Weight function used during prediction. With 'distance', closer neighbors of a query point will have a greater influence than neighbors which are further away. With 'uniform', all points in each neighborhood are weighted equally.	['uniform', 'distance']

The anomaly detection k-NN configuration search is similar to the forecasting one, including both the number of neighbors and the weight function in place.

Table 5.20: MLP hyperparameters' name, description and value interval being studied.

Hyperparameter	Description	Search space
`n_hidden_layers`	The number of hidden layers to include in the MLP architecture.	[1, 10]
`n_hidden_units`	The number of neurons to append to each hidden layer.	[1, 20 * n_features]
`max_iter`	The maximum number of iterations allowed during fitting.	[100, 3000]
`alpha`	The regularization (penalty) term that avoids overfitting by constraining weight size.	[1e-5, 1e-2]

The parameter domain for the anomaly detection MLP is equivalent to the forecasting version.

Table 5.21: SVM hyperparameters' name, description and value interval being studied.

Hyperparameter	Description	Search space
`C`	Penalty parameter C of the error term.	[0.0001, 100]
`kernel`	Specifies the kernel type to be used in the algorithm.	['rbf', 'poly', 'sigmoid']
`max_iter`	The maximum number of iterations to be run.	[100, 2000]

The SVM's kernel is an important factor in defining the data transformations during the hyperplane search. Since the linear SVM is being tested, all the remaining are of interest as well. Once again, the `C` is sampled from its linear counterpart.

5.5 Revision Phase

The Reuse phase of the CBR cycle gathers the chosen model, from the Retrieval Phase, and outputs its hyperparameters, prone to tuning. It is during the next phase, Revision, where the model is effectively built and tuned in relation to its hyperparameters' variation. The Revision phase is supported by Bayesian Optimization, detailed below.

5.5.1 Bayesian Optimization

The presented framework integrates bayesian optimization through `scikit-optimize`, an open-source library. `Scikit-optimize` is a `Python` package built on `NumPy`, `SciPy` and `Scikit-learn`, and specializes in minimizing expensive and noisy black-box functions such as the ones provided in this framework. The focus for this project is on bayesian optimization where the surrogate model is made up of GP[19], but the library also provides other surrogate options, such as decision trees[20] and gradient boosted trees[21].

This implementation of the optimizer expects as input the model itself, a dictionary structure containing the hyperparameters and their domain, the number of random start iterations, the number of calls to the surrogate model and an array of callback methods. This strict flow made it necessary to wrap the implemented models, as well as transformations on the time series, into a new structure that only outputs 1 single performance measure. Figure 5.9 shows the optimizer's process on the model wrapper structure.

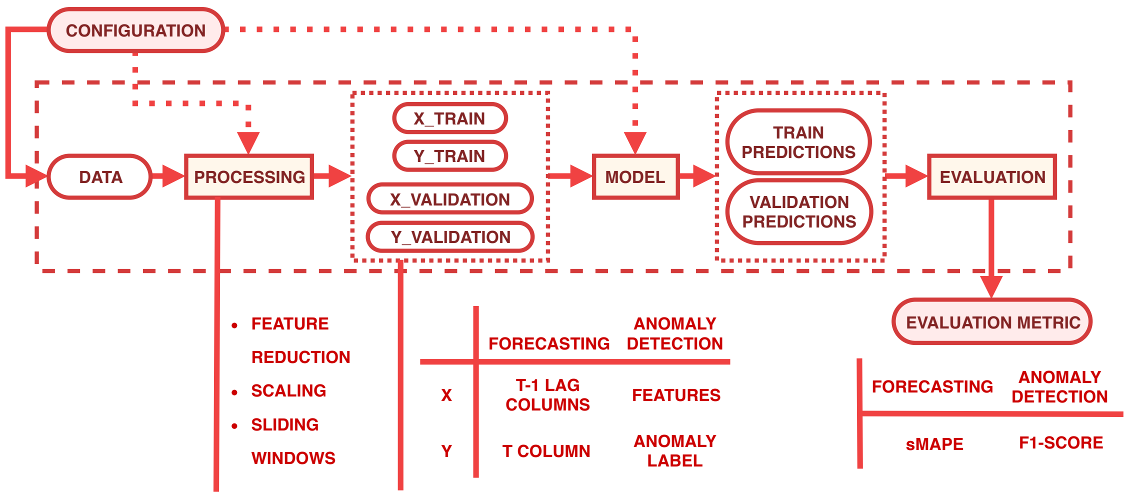


Figure 5.9: Model wrapper architecture used within the optimizer.

The model wrapper is responsible for manipulating the data and the model configuration, and then perform an evaluation on that configuration, finally outputting a single evaluation metric in use by the optimizer.

The **Configuration** is most important for the model, for it contains the hyperparameter values at each given iteration that may minimize the model's evaluation error. However, it also houses the factor for the sliding windows procedure, also referred to as the number of lags.

During **Processing**, the sliding windows procedure is first by feature reduction (mainly used in anomaly detection, where the features are not being estimated, unlike in forecasting) and then by data scaling (because some of the implemented models are sensitive to large values). The result from processing is the collection of train and validation splits used further during the model fitting

[19]`https://scikit-optimize.github.io/#skopt.gp_minimize`
[20]`https://scikit-optimize.github.io/#skopt.forest_minimize`
[21]`https://scikit-optimize.github.io/#skopt.gbrt_minimize`

and evaluation. In forecasting, the features are the generated $t-1$ lag columns, while for anomaly detection, sliding windows are avoided. The label associated with forecasting is the actual t column and, in anomaly detection, it contains a binary value (0 for non-anomaly, 1 for anomaly). The train split (`X_TRAIN`), along with its true labels (`Y_TRAIN`), is then fed to the **Model** for fitting.

From this point, the model is used to predict both the train and validation observations. These predictions are then passed over to **Evaluation**, producing the sMAPE metric, for forecasting, due to its easy interpretability, and F1-Score for anomaly detection, due to its balance between precision and recall, and its robustness to imbalanced data.

In order to define the number of iterations allowed by the optimizer, we accounted for the number of hyperparameters to modify, as well as the execution time for the implemented models. The MLP is the model with the most extensive hyperparameter search (a total of 4 variables), and the LSTM is one of the slowest performing options. Therefore, the optimizer was configured to initialize 20 random configurations and then proceed to optimize the configuration for 150 iterations. In any case, an early stop callback array was provided. This method checks if the evaluation metric did not decrease for 15 consecutive iterations, and stops the search. Once the optimizer has finished, the model is, once again, configurated, but with the optimal parameter configuration. The training split now also includes the validation split, and the test split is used for evaluation of the final model, as shown in Figure 5.10.

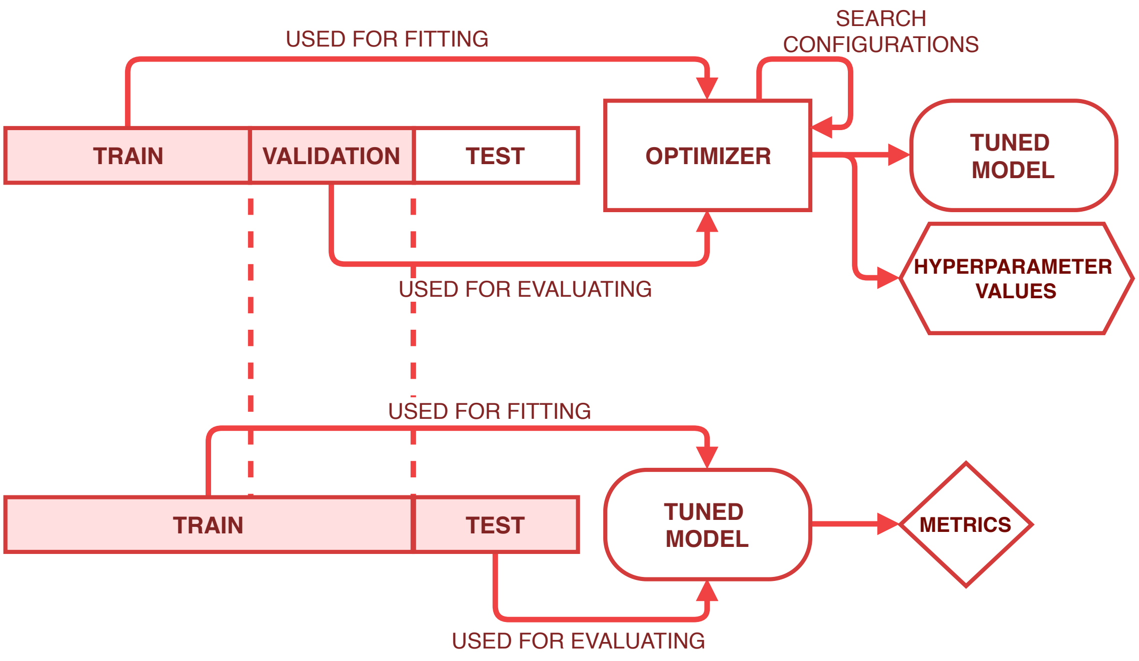


Figure 5.10: Dataset application in the optimizer and the final tuned model, where the red portions highlight the used splits.

The output provided by the Revision phase contains elements such as the solution model, its tuned parameters and the resulting metrics. It is during the Retain Phase that the solution model (the case label) as well as its configuration and metrics are appended to the case and stored for future comparisons. The Retain Phase is fundamental to CBR, since it can provide not only a similar solution, but also a degree of uncertainty associated with its integrating cases' metrics. Figures 5.11, 5.12 and 5.13 present additional graphical outputs detailing the accuracy of the tuned model during both training and testing phases.

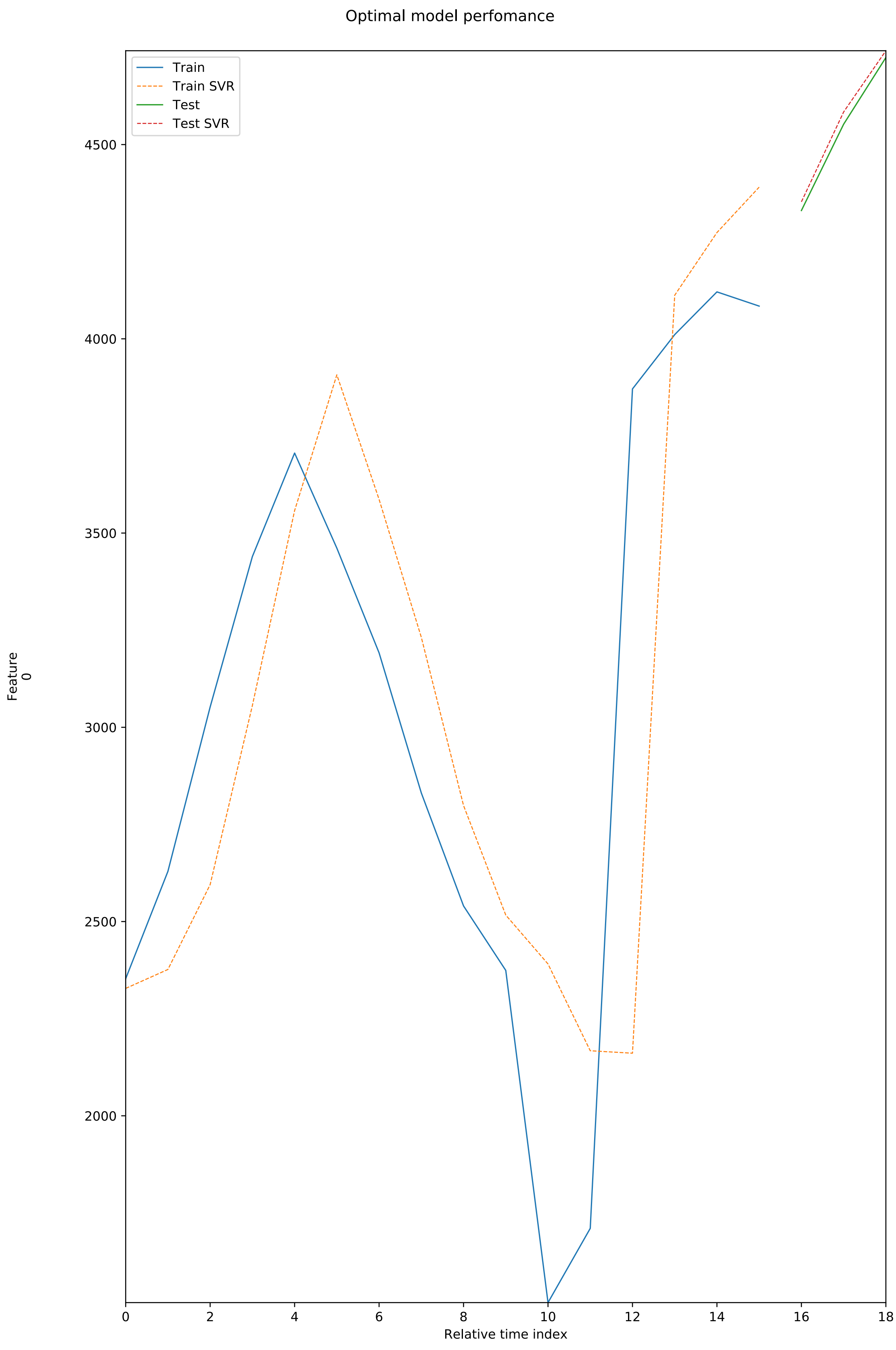


Figure 5.11: UTS Forecasting CBR graphic for time series N 14, from the M3-Competition, solved with SVR.

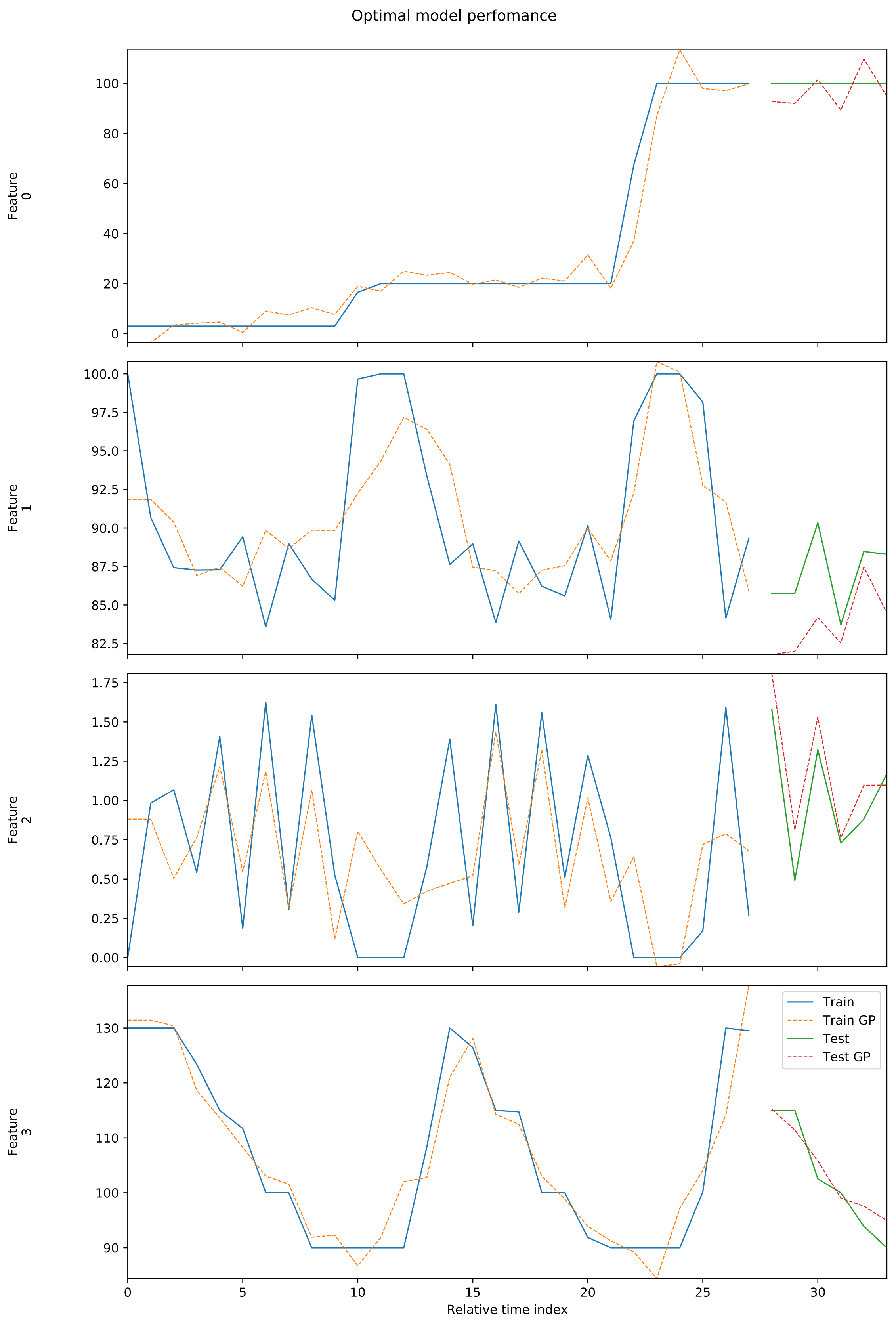


Figure 5.12: MTS Forecasting CBR graphic for the Hydraulics dataset, solved with GP.

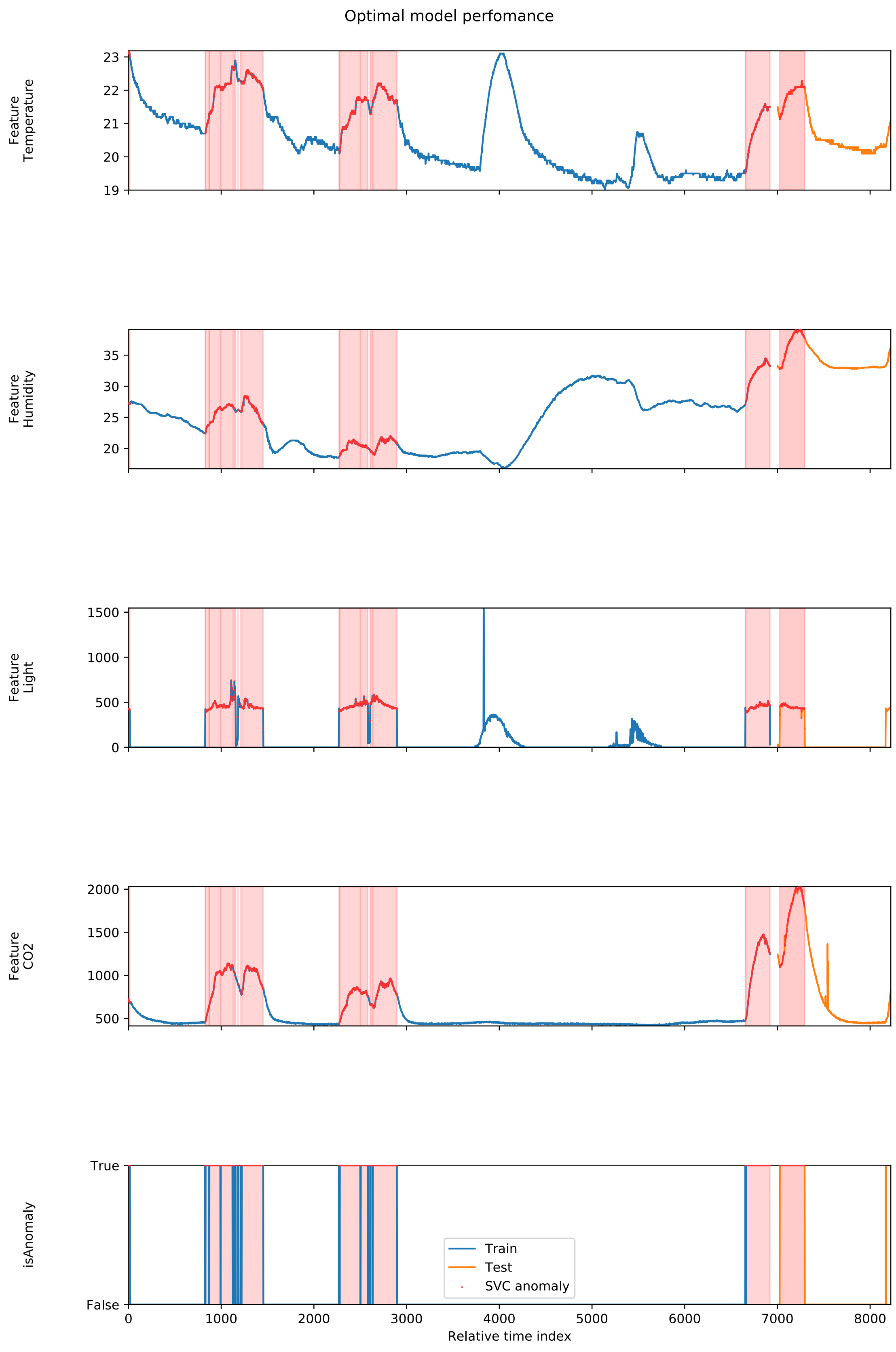


Figure 5.13: MTS Anomaly Detection CBR graphic for the Occupancy dataset, solved with SVM.

Chapter 6

Experimental Setup

At this point, a presentation of the approach and its setup have been made, but an evaluation of the approach is crucial in identifying the framework's capacity to:

- Assign an optimal Forecasting or Anomaly Detection model to a new dataset;
- Guarantee that the optimal model has the optimal hyperparameters.
- Produces accurate forecasts and accurate anomaly detections.

This chapter presents the pipelines in use to properly evaluate the Case-based Reasoning (CBR)-based classifications and a result analysis focusing on three major scenarios:

- Univariate Time Series (UTS) Forecasting CBR;
- Multivariate Time Series (MTS) Forecasting CBR;
- MTS Anomaly Detection CBR;

The result analysis for each scenario addresses the global performance of the CBR approach, further detailed into the performance per label (in this case, a model solution) and, finally, addressing the effect of the Bayesian Optimization.

6.1 Pipeline

UTS CBR classifications rely on a k-Nearest Neighbours (k-NN) that has been trained on a portion of the case-base and evaluated on the remainder. This concept is almost identical in the case of the MTS CBR, but the classifier in this scenario (Semi Metric Ensemble Time Series (SMETS)) is custom, and, therefore, not compatible with `Scikit-Learn`'s cross-validation process. MTS data is also not prone to feature operations (feature reduction and feature selection), as these are handled within SMETS. Once the model is fitted, predictions for the test split are calculated and later evaluated, resulting in the metrics described in Section 5.2.2. The intermediate results include a multiclass confusion matrices. The final results are expressed by a uniform average of all iterations, for all metrics. This entire process is repeated for 30 iterations, in conformity with the Central Limit Theorem [40].

Figures 6.1 and 6.2 depict the experimental setup where the results are provided and processed into a performance measure for the UTS and MTS cases, respectively.

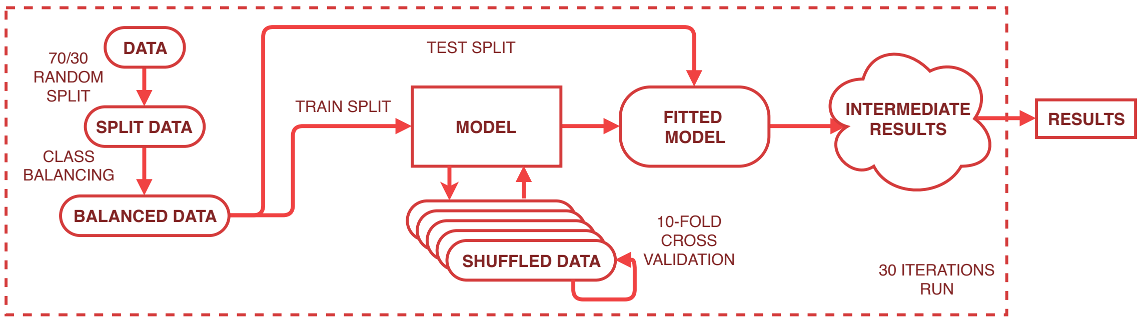


Figure 6.1: UTS CBR experimental setup.

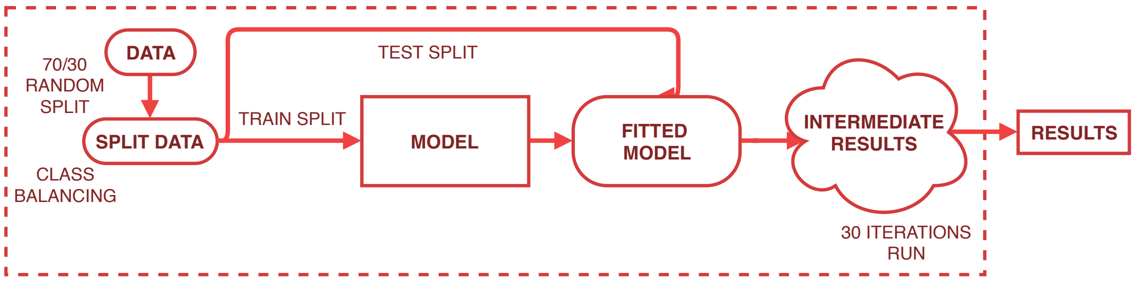


Figure 6.2: MTS CBR experimental setup.

6.2 Results Analysis

The previous pipelines, shown in Figures 6.1 and 6.2 were fundamental in order to collect metrics for the CBR and the Bayesian Optimization. In this section, these results are statistically arranged (using their mean and standard deviation) so that it's possible to assess the CBR approaches as well as the overall quality of the Bayesian Optimization, for each scenario. Section 6.2.1 details the UTS Forecasting CBR approach, while Section 6.2.2 handles MTS Forecasting and Anomaly Detection CBR.

6.2.1 UTS CBR

The UTS CBR rests on k-NN predictions to annotate a dataset with the possible best solution. For this scenario, a total of 2306 time series from the Makridakis Competition (M-Competition) were analyzed, resulting in the distribution shown in Figure 6.3, where the each time series was labeled with an optimal model as solution.

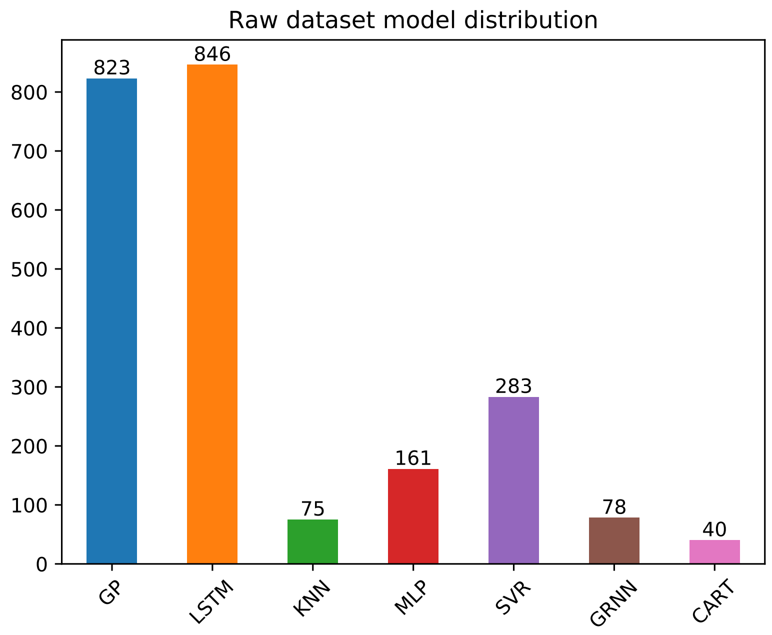


Figure 6.3: UTS Forecasting CBR dataset class distribution.

There is a large tendency towards Gaussian Process (GP) and Long Short Term Memory (LSTM), followed by Support Vector Regression (SVR) while the remaining models get less than 25% exposure. On one hand, these results differ from samples drawn from the M3-Competition [59], where the best performing algorithms included Multi Layer Perceptron (MLP) at the top. The fact that not all of MLP's hyperparameters were explored could have contributed to the model's low frequency. On the other hand, both GP and LSTM represent more than 75% of the dataset size. LSTMs do have the theoretical advantage of producing meaningful data relations in function of the time component [39]. 1921 of the evaluate time series are non-stationary, which can explain GP frequency, since GPs have also been successful, especially with non-stationary time series [7, 18]. While GP and LSTM appear to be more robust, given the different natures of the time series provided during the M-Competition, it is important to recall that no algorithm, nor pair of algorithms, is universally better as a solution, which is why this framework provides other models that are able to achieve good results.

Table 6.1: UTS Forecasting CBR results' metric distibution.

Metric	**Mean**	**Std. Dev**
Accuracy	0.38	0.01
Micro F1-score	0.38	0.01
Macro F1-score	0.21	0.02
Macro weighted F1-score	0.38	0.40
Micro Recall	0.38	0.01
Macro Recall	0.21	0.02
Macro weighted Recall	0.38	0.01
Micro Precision	0.38	0.01
Macro Precision	0.21	0.02
Macro weighted Precision	0.38	0.01

Table 6.1 presents all the calculated metrics for the CBR global performance, that is, averaging all classes' metrics. Due to the nature of the obtained dataset , the Macro weighted measures are favorable than the remaining. Note that the Macro weighted Recall and Precision output the same distribution, which means that the CBR has classified an equal amount of examples as False Positives and False Negatives. These metrics embed the F1-score formula, which states that when Precision and Recall are equal, the result is equal to either of them. Thus, the Macro weighted F1-score attains the same result, at 38%. However, it is slightly unstable, due to its high standard deviation. To better visualize per class performance, and help understand the effect of the data imbalance, Table 6.2 presents the metrics for each of the classes.

Table 6.2: UTS Forecasting CBR classes' metric distribution.

	Metrics							
	Accuracy		**Precision**		**Recall**		**F1-score**	
Model	**Mean**	**Std. Dev**	**Mean**	**Std. Dev**	**Mean**	**Std. Dev**	**Mean**	**Std. Dev**
GP	0.52	0.03	0.52	0.02	0.52	0.03	0.52	0.02
LSTM	0.43	0.03	0.45	0.02	0.43	0.03	0.44	0.02
KNN	0.15	0.06	0.14	0.05	0.15	0.06	0.14	0.05
MLP	0.10	0.04	0.10	0.04	0.10	0.04	0.10	0.04
SVR	0.16	0.03	0.15	0.03	0.16	0.03	0.16	0.03
GRNN	0.10	0.05	0.09	0.05	0.10	0.05	0.09	0.04
CART	0.03	0.05	0.03	0.06	0.03	0.05	0.03	0.05

In Table 6.2, it becomes clear that the overall presence of GP and LSTM examples contribute to a more accurate prediction by the CBR k-NN. All the collected metrics show GP obtaining the best results, followed closely by LSTM. While misclassifications occur, and account for almost half of the GP examples (better seen through Accuracy), this is still the class that is best predicted. This outcome is expected, considering these are the best represented classes in the dataset, while the remaining classes struggle with correct classifications.

When comparing k-NN and SVR, this turns into an irregular case, since SVR have almost 4 times more examples than k-NN, and yet results appear to favor them equally. However, looking at the standard deviation, k-NN appears more unstable in its performance, which could have resulted from k-NN misclassifications occurring more frequently than with SVR. A similar situation develops when comparing MLP with Generalized Regression Neural Network (GRNN), where the latter has approximately half the examples of the former and still manages to perform equally. These cases most likely result from the feature's discriminative power not being high enough with classes such as MLP and SVR, where the number of observations should positively influence the metrics.

As for Classification and Regression Trees (CART), the results are in tandem with the expected: this label has the lowest frequency in the raw dataset (1.7% of the entire dataset, 4.7% of the most common class, LSTM), which heavily compromises the UTS CBR's ability to learn it properly.

It's important to notice that while the CBR approach operates on an imbalanced dataset, the labels for each of its observation were given according to the best performing model for said observation. Table 6.3 displays the Symmetric Mean Absolute Percentage Error (sMAPE) distribution from the forecasting models on UTS CBR dataset.

Table 6.3: UTS Forecasting average sMAPE per model, and over all models.

Model	Mean	Std. Dev
CART	5.63 %	6.59 %
GP	1.93 %	3.85 %
GRNN	5.73 %	5.77 %
KNN	4.71 %	5.12 %
LSTM	3.46 %	5.22 %
MLP	4.03 %	4.57 %
SVR	3.98 %	4.74 %
All	3.17 %	4.83 %

The results show that, when chosen as the optimal model, after hyperparameter tuning, GP presented the lowest error overall, while GRNN presented the highest. These observations do not necessarily relate to the survey where these models were evaluated [59], but, at this time, no mention of optimization was made, whereas the results at present are lower than the ones published. When accounting for the maximum deviation, both CART and GRNN stand out for being the least reliable models, but still manage to maintain a sMAPE under 13%. It is assumed that these approaches are not the most adequate for time series forecasting, in the sense that they don't necessarily have the capacity to model these structures accurately. In general, all algorithms perform well, despite their deviations, resulting in sMAPE lower than 8%.

6.2.2 MTS CBR

MTS get their definition for containing more than one UTS in their structure, which renders these datasets extensive in dimensions. This property increases the model training time, which is limited during the period of the internship, resulting in the generated MTS CBR datasets having fewer examples than their UTS counterpart. Additionally, these datasets are not as frequent in the literature as UTS for Forecasting. In order to enrich the MTS CBR, the datasets should contain more examples. Nonetheless, the following sections present the results for both Forecasting and Anomaly Detection MTS CBR.

Forecasting

In the scenario of MTS Forecasting, a total of 15 time series where processed. Table 6.4 shows the class distribution for these datasets.

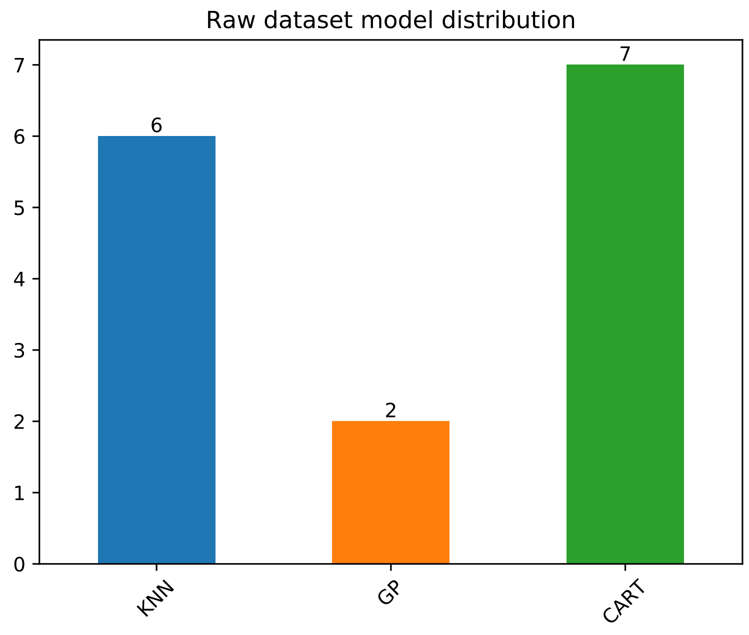


Figure 6.4: MTS Forecasting CBR dataset class distribution.

Do note that 10 of the 15 ran datasets stemmed from the same experimental setup, hence their optimal model would be the same. As a result, from the generation of the dataset resulted a strict distribution where most of the available models were not even selected. CART and k-NN were used with a similar frequency, while GP has less examples.

Table 6.4: MTS Forecasting CBR results' metric distibution.

Metric	**Mean**	**Std. Dev**
Accuracy	0.32	0.16
Micro F1-score	0.32	0.16
Macro F1-score	0.21	0.11
Macro weighted F1-score	0.29	0.16
Micro Recall	0.32	0.16
Macro Recall	0.26	0.14
Macro weighted Recall	0.32	0.16
Micro Precision	0.32	0.16
Macro Precision	0.21	0.11
Macro weighted Precision	0.30	0.19

As a result from a small set of examples which describe an imbalanced dataset, the MTS Forecasting CBR presents low performance metrics, as shown by Table 6.4. Using the Macro weighted F1-score as reference, given the imbalance in the data, this model performs, at best, with an F1-score of 0.45, which itself is not reliable. Additionally, not just for the Macro weighted F1-score, most metrics present a large deviation, which further shows how the lack of examples impacts this analysis. Nevertheless, the results for each specific class, shown in table 6.5, can provide more insight.

Table 6.5: MTS Forecasting CBR classes' metric distribution.

	Metrics							
	Accuracy		**Precision**		**Recall**		**F1-score**	
Model	**Mean**	**Std. Dev**	**Mean**	**Std. Dev**	**Mean**	**Std. Dev**	**Mean**	**Std. Dev**
GP	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
LSTM	-	-	-	-	-	-	-	-
KNN	0.19	0.32	0.15	0.25	0.19	0.32	0.15	0.22
MLP	-	-	-	-	-	-	-	-
SVR	-	-	-	-	-	-	-	-
GRNN	-	-	-	-	-	-	-	-
CART	0.56	0.37	0.44	0.29	0.56	0.37	0.46	0.28

In this scenario, the test split used for evaluation covers a total of 5 examples. Seeing as 4 of the 7 models are not even represented in the raw dataset, their values have been rendered as "-". Starting with GP, it is expected that the values all round down to 0.0 due to the existence of only 2 examples in the raw dataset. The only scenario where the CBR SMETS could have made a correct prediction would be if one of the examples went into the training split, and the remaining one into the test split, and even then, it's highly unlikely the model would make a correct prediction. Naturally, this model is badly represented and, therefore, so are its metrics.

For k-NN and CART, the same cannot be said. In fact, CART appears to have been the best learned label, with its F1-score reaching up to 0.46, on average. Still, the situation where k-NN has 1 example less than CART and performs much worse is irregular. It is speculated that the test split might have favored CART examples to k-NN, or that, due to 6 out of 10 of the Flight datasets being classified with CART, that the features were used to better describe this model type. During the M3-Competition [57], only UTS were used, but even then CART presented slightly better results than k-NN. In any case, both models have a gross standard deviation, which is ultimately the result of a short dataset.

The Bayesian Optimization results presented in Table 6.6 show that, each time each model was used, the error was acceptable.

Table 6.6: MTS Forecasting average sMAPE per model, and over all models.

Model	**Mean**	**Std. Dev**
CART	6.09 %	6.64 %
GP	9.63 %	5.61 %
GRNN	-	-
KNN	8.66 %	8.99 %
LSTM	-	-
MLP	-	-
SVR	-	-
All	7.59 %	7.69 %

The results tend to show that from the most to least frequent model, the average (and deviation) sMAPE increased, save for GP which was used only twice. Still, at worst, its maximum sMAPE was below 15.5%. Note that there are less than 10 examples per class, which may explain why all models possess such a high deviation. Overall, the average sMAPE never increased beyond 15.28%.

Anomaly Detection

In the case of Anomaly Detection, the dataset contains a total of 22 examples. Table 6.5 provides the class distribution within this dataset.

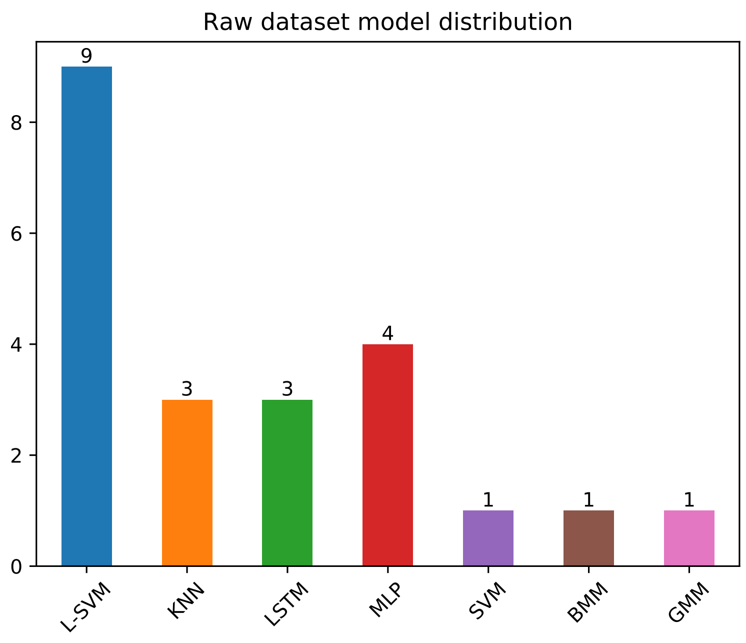


Figure 6.5: MTS Anomaly Detection CBR dataset class distribution.

During the generation of the dataset, the Linear Support Vector Machine (SVM) stood out, with 9 examples, while the remaining models were never selected more than 4 times. Alas, the number of examples is insufficient to properly evaluate the class distribution.

Table 6.7: MTS Anomaly Detection CBR results' metric distibution.

Metric	**Mean**	**Std. Dev**
Accuracy	0.1	0.08
Micro F1-score	0.1	0.08
Macro F1-score	0.05	0.04
Macro weighted F1-score	0.1	0.23
Micro Recall	0.1	0.08
Macro Recall	0.05	0.05
Macro weighted Recall	0.1	0.08
Micro Precision	0.1	0.08
Macro Precision	0.06	0.06
Macro weighted Precision	0.15	0.17

The lack of examples severely impacts the MTS CBR classifier, as shown by the metrics in Table 6.7. Given the imbalanced dataset, the macro weighted F1-score serves as a reference for results analysis. This result suggests that, for most iterations, the score is very low, while containing some examples where classification rose. Regarding the distribution, this accounted for a low mean, with a very high deviation. To help understand the CBR distribution, Table 6.8 highlights each class' performance.

Table 6.8: MTS Anomaly Detection CBR classes' metric distribution.

	Metrics							
	Accuracy		**Precision**		**Recall**		**F1-score**	
Model	**Mean**	**Std. Dev**	**Mean**	**Std. Dev**	**Mean**	**Std. Dev**	**Mean**	**Std. Dev**
L-SVM	0.19	0.2	0.26	0.31	0.19	0.2	0.19	0.19
KNN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
LSTM	0.0	0.0	0.03	0.11	0.03	0.12	0.03	0.11
MLP	0.0	0.0	0.03	0.18	0.03	0.18	0.03	0.18
SVM	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BMM	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
BGMM	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

The test split used for evaluation of the CBR entails 30% of the original data, which accounts for 6 examples. With the labels SVM, Bayesian Gaussian Mixture Model (BGMM) and Gaussian Mixture Model (GMM) represented only once, their metrics have been rendered null. The case of the k-NN is also particular, due to having the same number of observations as the LSTM and still rendering no positive values, for any metric. This is most likely due to the CBR not properly learning this type of example, thus there were no True Positives. However, cases such as LSTM and MLP present a slightly different behavior. According to the Precision, Recall and F1-score, these models had visibility, albeit minor, due to their inferior frequency, in relation to SVM with linear kernel. However, their Accuracy is stalled at 0, most likely due to the CBR not predicting them correctly significant times. Other than that, the linear SVM, which is the class with most observations, is the best inferred by the CBR. Note that the F1-score for all non-zero models oscillates aggressively. This further shows that misclassifications occur very frequently and that this CBR is underfitted.

However, when addressing each applied class directly, the results shown in Table 6.9 show promise.

Table 6.9: Anomaly Detection average F1-score per model, and over all models.

Model	**Mean**	**Std. Dev**
BGMM	0.61	0.0
GMM	0.64	0.0
KNN	0.8	0.05
L-SVM	0.93	0.05
LSTM	0.9	<0.01
MLP	0.89	0.11
SVM	0.92	0.0
All	0.87	0.11

Regarding the optimized models' performance, SVM, GMM and BGMM have a 0 standard deviation due to accounting for 1 example each. While Linear SVM are the most common, they also present, on average, the best score. However, this model's standard deviation is more irregular than the LSTM's, which stands out for an almost constant performance. Overall, the tuned models output a Macro weighted F1-score of 0.87, thus, optimization positively affected these models.

Chapter 7

Methodology and Planning

This section presents the methodology adopted for this project as well as the initial proposed plan and modifications that have been made during the first semester.

7.1 Methodology

The internship took place at Novabase, and, as such, followed the development methodology in practice at the company. In this case, the methodology in use was Scrum[1]. *Scrum* is an iterative and incremental methodology, used in *Agile Software Development*[2], by small teams, which contains a set of roles and practices that can be adjusted to the environment of each team it integrates. A key principle in *Scrum*, also present in the *Agile Manifesto*, is that requirements change throughout the timeline of a software project. Unlike traditional methodologies, this means that unpredictable changes can be managed easily. This methodology uses an empirical approach and accepts that a problem scope cannot be fully understood, let alone defined, rather focusing on quickly reacting to emergent challenges.

The development process in *Scrum* evolves by iterations, the *Sprints*, which usually last from two to four weeks. Each *Sprint*'s tasks, the *Sprint Backlog*, are determined in the *Sprint Planning Meeting*, chosen from a list of requirements to develop, prioritized, the *Product Backlog*. Requirements defined in the *Sprint Backlog* cannot change during the *Sprint*, which itself must always end at the defined time. Should a requirement not have been fulfilled in a *Sprint*, it goes back to the *Product Backlog*. Progress on a *Sprint* is supported by the use of *Burn Down Charts*, which monitor the time expenses associated with the tasks.

Progress is also discussed in daily meetings, which usually last 15 minutes, called *Daily Scrum Meeting*. In these, each team member presents finished tasks, the tasks to finish at that day and possible impediments they're facing. When a *Sprint* finishes, the team reviews the work and presents it to the *stakeholders* in a *Sprint Review Meeting*. To conclude, a *Sprint Retrospective Meeting* allows the team to reflect on the last *Sprint* and suggest changes where deemed necessary.

The major roles defined in *Scrum* are as follows.

Product Owner Responsible for maximizing the value of the product resulting from the work of the Development Team. Solely the *Product Owner* manages the *Product Backlog* (including expressing product backlog items and ordering items to maximize goals achievement).

Scrum Master Responsible for promoting and supporting *Scrum* by helping every role understand *Scrum* theory, practices, rules and values. The *Scrum Master* also provides services to both the *Product Owner* (finding techniques for effective *Product Backlog* management, fa-

[1] `https://ieeexplore.ieee.org/document/854065`
[2] `https://agilemanifesto.org/principles.html`

cilitating scrum events, etc) and the Development Team (coaching the developers, removing impediments to the development progress).

Team Consisting of members who deliver increments of the product at the end of each *Sprint*. The team does not recognize any titles for its members, regardless of the work being performed by each person.

For this project, the team was composed of Daniel Moura (*Product Owner*), Miguel Oliveira (*Scrum Master*) and Pedro Costa (*Team*). This internships ensued in the context of a solution that had no direct integration with any part of Novabase's solutions, thus, its development rested solely on the aforementioned people. Sprints were projected to have a duration of two weeks.

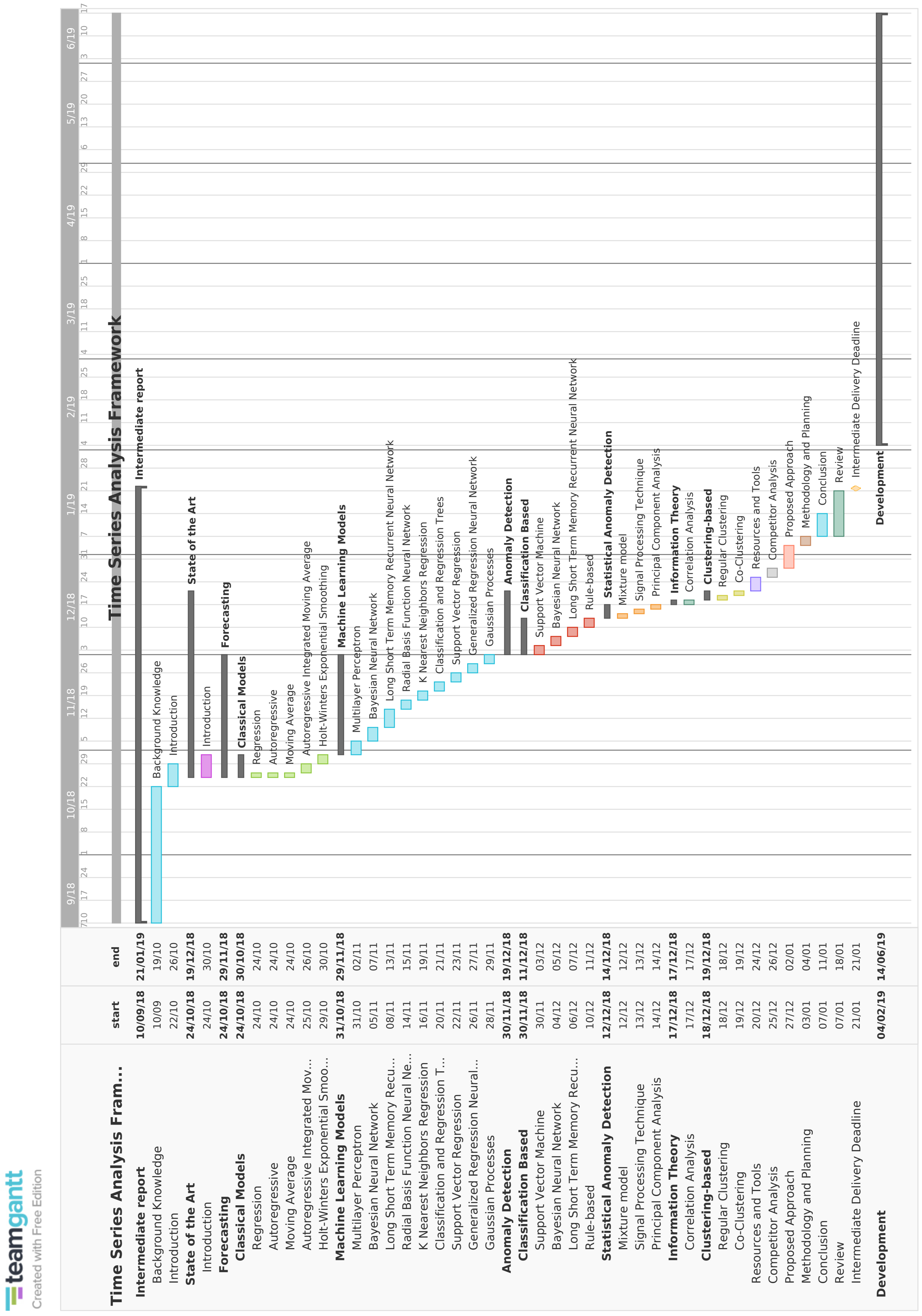
The Scrum process was implemented using Jira[3]. The main communication channel was Slack[4]. Project artifacts, which include source code and documentation, produced throughout the internship are stored and versioned using repositories hosted on BitBucket[5].

7.2 Planning

This section provides information on the project planning regarding both the first and second semester. An initial Gantt diagram is presented in Figure 7.1 and 7.2. However, given how some parts of the work carried out during the first semester exceeded the initial time estimate, Figure 7.3 shows the Gantt diagram for the actual time expenses regarding the first semester. Looking closely, one can see that Signal Processing Technique and Correlation Analysis have been removed from the Anomaly Detection state of the art approaches. While they were studied, their weight in the state of the art was not appear significative. In the case of the Information Theory Section, 2.2.3, a use case for the introduction was provided, thus this section was kept. However, Anomaly Detection proved an area where its problems are difficult to solve, effectively shifting the duration for this task up to January 4th, while the remainder of the tasks also got shifted. The last modification to the original schedule was made at Department of Informatics Engineering (DEI), where the delivery date was pushed from January 21st to January 23rd. In any case, the first semester was mainly used for information search on the internship area, regarding state of the art, tools, competitors and finally the proposal of the approach.

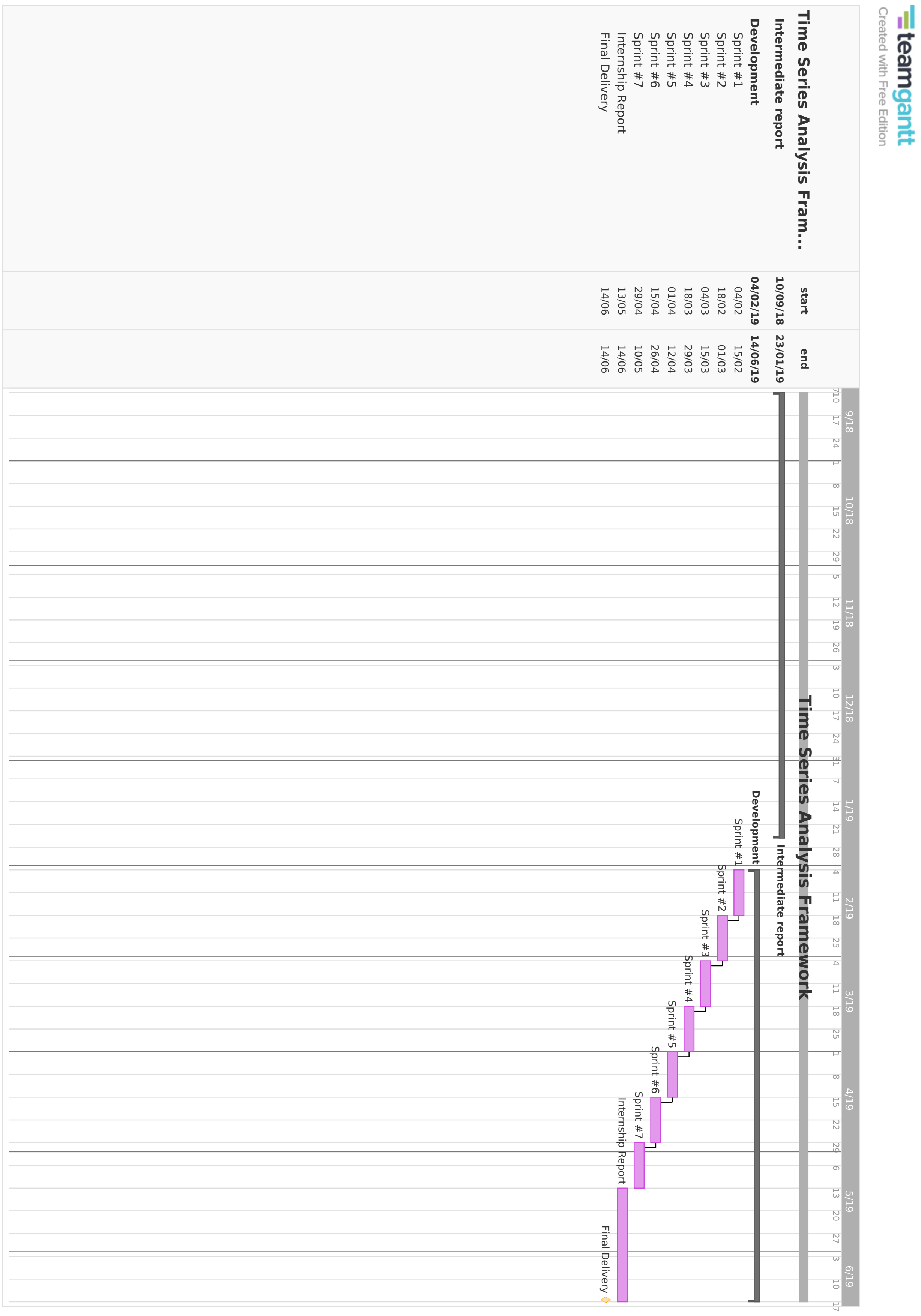
The second semester addressed the development of the project, which is divided into *sprints*, in accordance to the methodology in use.

[3]`https://www.atlassian.com/software/jira`
[4]`https://slack.com/`
[5]`https://bitbucket.org/`



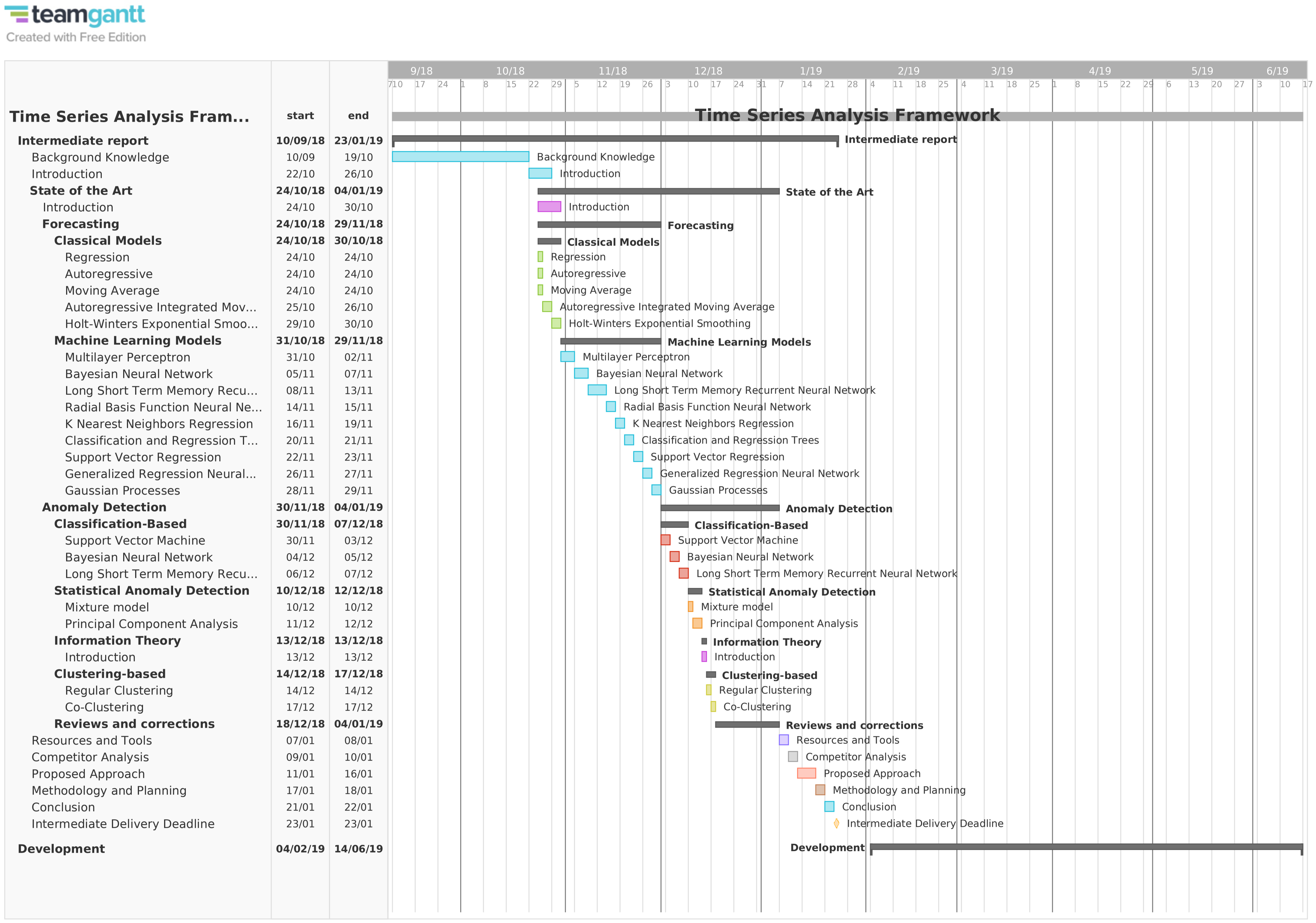
Time Series Analysis Fram...	start	end
Intermediate report	**10/09/18**	**21/01/19**
Background Knowledge	10/09	19/10
Introduction	22/10	26/10
State of the Art	**24/10/18**	**19/12/18**
Introduction	24/10	30/10
Forecasting	**24/10/18**	**29/11/18**
Classical Models	**24/10/18**	**30/10/18**
Regression	24/10	24/10
Autoregressive	24/10	24/10
Moving Average	24/10	24/10
Autoregressive Integrated Mov...	25/10	26/10
Holt-Winters Exponential Smoo...	29/10	30/10
Machine Learning Models	**31/10/18**	**29/11/18**
Multilayer Perceptron	31/10	02/11
Bayesian Neural Network	05/11	07/11
Long Short Term Memory Recu...	08/11	13/11
Radial Basis Function Neural Ne...	14/11	15/11
K Nearest Neighbors Regression	16/11	19/11
Classification and Regression T...	20/11	21/11
Support Vector Regression	22/11	23/11
Generalized Regression Neural...	26/11	27/11
Gaussian Processes	28/11	29/11
Anomaly Detection	**30/11/18**	**19/12/18**
Classification Based	**30/11/18**	**11/12/18**
Support Vector Machine	30/11	03/12
Bayesian Neural Network	04/12	05/12
Long Short Term Memory Recu...	06/12	07/12
Rule-based	10/12	11/12
Statistical Anomaly Detection	**12/12/18**	**14/12/18**
Mixture model	12/12	12/12
Signal Processing Technique	13/12	13/12
Principal Component Analysis	14/12	14/12
Information Theory	**17/12/18**	**17/12/18**
Correlation Analysis	17/12	17/12
Clustering-based	**18/12/18**	**19/12/18**
Regular Clustering	18/12	18/12
Co-Clustering	19/12	19/12
Resources and Tools	20/12	24/12
Competitor Analysis	25/12	26/12
Proposed Approach	27/12	02/01
Methodology and Planning	03/01	04/01
Conclusion	07/01	11/01
Review	07/01	18/01
Intermediate Delivery Deadline	21/01	21/01
Development	**04/02/19**	**14/06/19**

Figure 7.1: Initial Gantt Diagram detailing the first Semester.



Time Series Analysis Fram...	start	end
Intermediate report	**10/09/18**	**23/01/19**
Development	**04/02/19**	**14/06/19**
Sprint #1	04/02	15/02
Sprint #2	18/02	01/03
Sprint #3	04/03	15/03
Sprint #4	18/03	29/03
Sprint #5	01/04	12/04
Sprint #6	15/04	26/04
Sprint #7	29/04	10/05
Internship Report	13/05	14/06
Final Delivery	14/06	14/06

Figure 7.2: Initial Gantt Diagram detailing the second Semester.



Time Series Analysis Fram...	start	end
Intermediate report	**10/09/18**	**23/01/19**
Background Knowledge	10/09	19/10
Introduction	22/10	26/10
State of the Art	**24/10/18**	**04/01/19**
Introduction	24/10	30/10
Forecasting	**24/10/18**	**29/11/18**
Classical Models	**24/10/18**	**30/10/18**
Regression	24/10	24/10
Autoregressive	24/10	24/10
Moving Average	24/10	24/10
Autoregressive Integrated Mov...	25/10	26/10
Holt-Winters Exponential Smoo...	29/10	30/10
Machine Learning Models	**31/10/18**	**29/11/18**
Multilayer Perceptron	31/10	02/11
Bayesian Neural Network	05/11	07/11
Long Short Term Memory Recu...	08/11	13/11
Radial Basis Function Neural Ne...	14/11	15/11
K Nearest Neighbors Regression	16/11	19/11
Classification and Regression T...	20/11	21/11
Support Vector Regression	22/11	23/11
Generalized Regression Neural...	26/11	27/11
Gaussian Processes	28/11	29/11
Anomaly Detection	**30/11/18**	**04/01/19**
Classification-Based	**30/11/18**	**07/12/18**
Support Vector Machine	30/11	03/12
Bayesian Neural Network	04/12	05/12
Long Short Term Memory Recu...	06/12	07/12
Statistical Anomaly Detection	**10/12/18**	**12/12/18**
Mixture model	10/12	10/12
Principal Component Analysis	11/12	12/12
Information Theory	**13/12/18**	**13/12/18**
Introduction	13/12	13/12
Clustering-based	**14/12/18**	**17/12/18**
Regular Clustering	14/12	14/12
Co-Clustering	17/12	17/12
Reviews and corrections	**18/12/18**	**04/01/19**
Resources and Tools	07/01	08/01
Competitor Analysis	09/01	10/01
Proposed Approach	11/01	16/01
Methodology and Planning	17/01	18/01
Conclusion	21/01	22/01
Intermediate Delivery Deadline	23/01	23/01
Development	**04/02/19**	**14/06/19**

Figure 7.3: Initial Gantt Diagram detailing the second Semester.

Chapter 8

Conclusions

The selection of the best algorithm to produce either forecasts or anomaly detections on time series structures can be a complex task. The work carried out through the course of this internship tackled this important aspect when performing analysis on time series, namely, in regards to the most appropriate model selection, as well as the model's hyperparameter tuning, commonly known as Auto-ML. To achieve this, a selection of Univariate Time Series (UTS) features was defined and taken into account by autonomous selectors built on the Case-based Reasoning (CBR) paradigm. In the case of Multivariate Time Series (MTS), a semi-metric, Semi Metric Ensemble Time Series (SMETS) was integrated and adapted to overcome some of its limitations, while providing a more flexible interface for comparing differents MTSs.

The State of the Art shows that Machine Learning (ML) approaches have been very frequent in the literature and provide excellent results in the domain of forecasting and anomaly detection, making them a viable option when compared to classical options and when considering the methods to include in the presented architecture for the proposed approach. Nevertheless, the referred classical models are even now a staple in time series forecasting, given their high accuracy and appropriate complexity. With Bayesian Optimization, it was expected that the implemented approaches' results quality increased, which was confirmed for the UTS scenario. Regarding the MTS scenarios, the final assessments showed that while there was no comparison with the State of the Art, these CBR models maintained a low error.

The Competitor Analysis shows that time series analysis contains a series of tasks that appear frequently in the business domain, having been explored by both companies who sought out to best market this service, as well as more popular competitors, such as Amazon and Microsoft. Current solutions also provide ML algorithms to cater the aforementioned services, sometimes even employing libraries that are available and were described as resources and tools that may as well be used in the content of this work. However, when it came to the presence of an automated model manufacturing, only two competitors stood out for employing Auto-ML, and even then, it is not sure what optimization method they use, whilst in this work, the state of art regarding hyperparameter tuning was integrated.

Once a sensitive analysis on the background knowledge, the state of the art and applications of the state of the art were collected, an approach based on CBR and Bayesian Optimization was presented and developed. This approach considered a total of 3 scenarios to attend to: UTS Forecasting, MTS forecasting and MTS Anomaly Detection. All of these scenarios warranted datasets retrieved from both the Makridakis Competitions (M-Competitions) and the University of California Irvine (UCI) Repository, which led to an analysis of the best performing models, in order to obtain a dataset which could be used to fit each scenario's CBR. In order to assess the proposed architecture, the main components (Model Selection and Model Hyperparameter Tuning) were evaluated in their ability to perform optimal model assignments and to reduce the errors associated with the practice of forecast and anomaly detections.

The results showed that the most sought out scenarios (MTS Forecasting and Anomaly Detection) were also the ones facing more obstacles, specifically dataset collection. Due to a shortage in examples for the CBR approaches, these models failed to produce meaningful results (Macro weighted F1-score of 0.29 for Forecasting MTS and of 0.1 for Anomaly Detection MTS) and should be addressed with priority in a future iteration. However, the same did not ensue for UTS Forecasting, where over 2000 examples were processed, and provided a fitted CBR with a better predictive power than either MTS CBR counterparts, scoring an averaged 0.38 Macro weighted F1-score.

8.1 Future Work

It is important to acknowledge the work carried out during internship, by accounting for the implemented requirements as well as less linear ventures, such as the adaptation of the SMETS for pairs of MTS with different number of observations. Nevertheless, not all requirements were fulfilled nor as much thought went into refining the framework. Thus, this section aggregates a study on how the work can be continuously enhanced, regarding both the phases of model selection and model hyperparameter tuning. Remarks about the architecture are also detailed. Starting with the datasets, transformations applied to them and metrics, the following topics were gathered:

- UTS datasets are compared through the k-Nearest Neighbours (k-NN) algorithm, which needs a feature vector in order to make a prediction. Currently, this feature vector is computed by external library TSFRESH, and while this approach presents good results, feature engineering in itself could be further specificied in relation to time series;
- The UTS Forecasting CBR was trained on a total of 2306 time series stemming from the M3-Competition, but resorting to other iterations of the M-Competition would enrich the CBR dataset. The same suggestion is even more important in both of the MTS scenarios, where the presented results suffer heavily from a short list of examples;
- SMETS itself can be further explored regarding the way it handles uneven number of observations: Instead of resorting to under or over sampling, unpaired observations could be penalized in the same fashion as features;
- The developed framework has the purpose of integrating other Novabase products, which demand real-time forecasts and anomaly detection, but SMETS has more latency than the k-NN in use for UTS CBR. As such, in order to maximize performance, only a fraction of the datasets could be considered for drawing comparisons;
- The approaches integrated into the Forecasting Selector were also prone to evaluation, specifically through Symmetric Mean Absolute Percentage Error (sMAPE) interpretation, due to it allowing direct comparisons between different scaled time series, but it also carries problems in regards to its assymetric penalization. This issue demands a more sensitive research to find a more appropriate measure for performance.

After model selection starts the model hyperparameter tuning (Bayesian Optimization) which garnered the following conclusions:

- The more complex models have many hyperparameters that can influence the optimization and, consequently, the results. Refining the hyperparameter domains and including more which are part of some models' architecture increases the search space but can benefit the quality and/or performance of the results;
- `scikit-optimize` is the library from which Bayesian Optimization is being drawn, and it can implement more surrogate models than just Gaussian Process (GP). A study focused on the effects of this component could provide solutions better suited to meet the user's needs in regards to efficiency;
- An initial approach considered for Forecasting, Bayesian Neural Network (BNN), was ultimately removed due to difficulties during the integration process, namely inconsistencies in the output predictions for MTSs, which lead to erroneous results. However, this approach is frequent in Forecasting literature, where it provides good results, so an effort to include at least this model should be made;

- Approaches such as Long Short Term Memory (LSTM) already have flexibility in the way their architecture is generated, albeit fixed during the internship, but other Neural Network (NN) approaches, such as Multi Layer Perceptron (MLP) and BNN could also be adapted to be expressed in different architectures, which could improve efficiency of the fitted models.

Considering what concerns the actual architecture, requirement PR.01, listed in Table 4.7, was not implemented, due to its lower priority. It is worth discussing the existence of a time threshold that supports real-time Forecasting and Anomaly Detection, due to the products this framework is to integrate, by limiting the model choices and their tuning time, at the cost of quality. Thus, the framework would benefit from two tiers: One focused on quality and another focused on execution speed. In the scope of the framework, this requirement has a lower priority in relation to other functional requirements, but it is imperative if the framework should integrate Novabase services. Additionally, in order to provide the framework to several remote machines without the need for several local instances, the envelopment of the framework in a Representational State Transfer (REST) Application Programming Interface (API) is also important.

References

[1] (2003). Nist/sematech e-handbook of statistical methods. section 6.4 introduction to time series analysis.

[2] Adhikari, R. and Agrawal, R. (2013). An introductory study on time series modeling and forecasting. *arXiv preprint arXiv:1302.6613*.

[3] Ahmed, M. (2014). Network traffic pattern analysis using improved information theoretic co-clustering based collective anomaly detection.

[4] Ahmed, M., Mahmood, A. N., and Hu, J. (2016a). A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19 – 31.

[5] Ahmed, M., Mahmood, A. N., and Islam, M. R. (2016b). A survey of anomaly detection techniques in financial domain. *Future Generation Computer Systems*, 55:278 – 288.

[6] Ahmed, N. K., Atiya, A. F., Gayar, N. E., and El-Shishiny, H. (2010a). An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29(5-6):594–621.

[7] Ahmed, N. K., Atiya, A. F., Gayar, N. E., and El-Shishiny, H. (2010b). An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29(5-6):594–621.

[8] Alon, I., Qi, M., and Sadowski, R. J. (2001). Forecasting aggregate retail sales:: a comparison of artificial neural networks and traditional methods. *Journal of Retailing and Consumer Services*, 8(3):147–156.

[9] Armstrong, J. S. (2001). *Principles of forecasting: a handbook for researchers and practitioners*, volume 30. Springer Science & Business Media.

[10] Arshad, F. M., Ghaffar, R. A., et al. (1986). *Crude Palm Oil Price Forecasting Box-Jenkins Approach*. Universiti Pertanian Malaysia.

[11] Bailey, K. (2016). Gaussian processes for dummies.

[12] Basu, S., Bilenko, M., and Mooney, R. J. (2004). A probabilistic framework for semi-supervised clustering. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 59–68, New York, NY, USA. ACM.

[13] Bauer, M. (1995). *General regression neural network, GRNN: A neural network for technical use*. University of Wisconsin–Madison.

[14] Ben-Gal, I. (2008). Bayesian networks. *Encyclopedia of statistics in quality and reliability*, 1.

[15] Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*.

[16] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA. ACM.

[17] Box, G. and Jenkins, G. (1976). *Time series analysis: forecasting and control*. Holden-Day series in time series analysis. Holden-Day.

[18] Brahim-Belhouari, S. and Bermak, A. (2004). Gaussian process for nonstationary time series prediction. *Computational Statistics and Data Analysis*, 47(4):705 – 712.

[19] Breiman, L. (1984). *Classification and regression trees*. Routledge.

[20] Brochu, E., Cora, V. M., and de Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599.

[21] Brockwell, P. J., Davis, R. A., and Calder, M. V. (2002). *Introduction to time series and forecasting*, volume 2. Springer.

[22] Broomhead, D. S. and Lowe, D. (1988). Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, Royal Signals and Radar Establishment Malvern (United Kingdom).

[23] Chatfield, C. (2003). *The Analysis of Time Series: An Introduction, Sixth Edition*. CRC press.

[24] Chauhan, S. and Vig, L. (2015). Anomaly detection in ecg time signals via deep long short-term memory networks. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–7. IEEE.

[25] Chen, C., Twycross, J., and Garibaldi, J. M. (2017). A new accuracy measure based on bounded relative error for time series forecasting. *PloS one*, 12(3):e0174202.

[26] De Mantaras, R. L., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M. L., T COX, M., Forbus, K., et al. (2005). Retrieval, reuse, revision and retention in case-based reasoning. *The Knowledge Engineering Review*, 20(3):215–240.

[27] Dorvlo, A. S., Jervase, J. A., and Al-Lawati, A. (2002). Solar radiation estimation using artificial neural networks. *Applied Energy*, 71(4):307–319.

[28] Drucker, H., Burges, C. J., Kaufman, L., Smola, A. J., and Vapnik, V. (1997). Support vector regression machines. In *Advances in neural information processing systems*, pages 155–161.

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

[30] Eskin, E. (2000). Anomaly detection over noisy data using learned probability distributions. In *In Proceedings of the International Conference on Machine Learning*. Citeseer.

[31] Eskin, E., Arnold, A., Prerau, M., Portnoy, L., and Stolfo, S. (2002). A geometric framework for unsupervised anomaly detection. In *Applications of data mining in computer security*, pages 77–101. Springer.

[32] Faraway, J. and Chatfield, C. (1995). Time series forecasting with neural networks: A case study. *University of Bath, Bath (United Kingdom), Research Report*, pages 95–06.

[33] G. Brown, R. and D.Litte, A. (1956). Exponential smoothing for predicting demand.

[34] González, J. (2017). Introduction to bayesian optimization. `http://gpss.cc/gpmc17/slides/LancasterMasterclass_1.pdf`.

[35] Goodwin, P. and Lawton, R. (1999). On the asymmetry of the symmetric mape. *International Journal of Forecasting*, 15(4):405 – 408.

[36] Grosse R., S. N. (2015). Mixture models.

[37] Heller, K. A., Svore, K. M., Keromytis, A. D., and Stolfo, S. J. (2003). One class support vector machines for detecting anomalous windows registry accesses. In *Proc. of the workshop on Data Mining for Computer Security*, volume 9.

[38] Hill, T., O'Connor, M., and Remus, W. (1996). Neural network models for time series forecasts. *Management Science*, 42:1082–1092.

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

[40] Hogg, R. V., Tanis, E. A., and Zimmerman, D. L. (1977). *Probability and statistical inference*, volume 993. Macmillan New York.

[41] Hu, W., Liao, Y., and Vemuri, V. R. (2003). Robust anomaly detection using support vector machines. In *Proceedings of the international conference on machine learning*, pages 282–289.

[42] Hyndman, R. J. and Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, 22(4):679 – 688.

[43] J. Willmott, C. and Matsuura, K. (2005). Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate Research*, 30:79.

[44] Jolliffe, I. (2011). Principal component analysis. In *International encyclopedia of statistical science*, pages 1094–1096. Springer.

[45] Kalekar, P. S. (2004). Time series forecasting using holt-winters exponential smoothing. *Kanwal Rekhi School of Information Technology*, 4329008:1–13.

[46] Kanchymalay, K., Salim, N., Sukprasert, A., Krishnan, R., and Hashim, U. R. (2017). Multivariate time series forecasting of crude palm oil price using machine learning techniques. In *IOP Conference Series: Materials Science and Engineering*, volume 226, page 012117. IOP Publishing.

[47] Kolodner, J. (1992). An introduction to case-based reasoning. *Artificial Intelligence Review*, 6:3–34.

[48] Kotsialos, A., Papageorgiou, M., and Poulimenos, A. (2004). Long-term sales forecasting using holt–winters and neural network methods. *Journal of Forecasting*, 24(5):353–368.

[49] Kruegel, C., Mutz, D., Robertson, W., and Valeur, F. (2003). Bayesian event classification for intrusion detection. In *null*, page 14. IEEE.

[50] Lane, D. (2003). Online statistics education: A multimedia course of study. In *EdMedia: World Conference on Educational Media and Technology*, pages 1317–1320. Association for the Advancement of Computing in Education (AACE).

[51] Laptev, N., Yosinski, J., Li, L. E., and Smyl, S. (2017). Time-series extreme event forecasting with neural networks at uber. In *International Conference on Machine Learning*, number 34, pages 1–5.

[52] Lazzaroni, M., Ferrari, S., Piuri, V., Salman, A., Cristaldi, L., and Faifer, M. (2015). Models for solar radiation prediction based on different measurement sites. *Measurement*, 63:346 – 363.

[53] Lee, W. and Xiang, D. (2001). Information-theoretic measures for anomaly detection. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S P 2001*, pages 130–143.

[54] Loh, W.-Y. (2011). Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23.

[55] Makridakis, S., Andersen, A., Carbone, R., Fildes, R., Hibon, M., Lewandowski, R., Newton, J., Parzen, E., and Winkler, R. (1982). The accuracy of extrapolation (time series) methods: Results of a forecasting competition. *Journal of forecasting*, 1(2):111–153.

[56] Makridakis, S., Chatfield, C., Hibon, M., Lawrence, M., Mills, T., Ord, K., and Simmons, L. F. (1993). The m2-competition: A real-time judgmentally based forecasting study. *International Journal of Forecasting*, 9(1):5 – 22.

[57] Makridakis, S. and Hibon, M. (2000). The m3-competition: results, conclusions and implications. *International journal of forecasting*, 16(4):451–476.

[58] Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2018a). The m4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*.

[59] Makridakis, S., Spiliotis, E., and Assimakopoulos, V. (2018b). Statistical and machine learning forecasting methods: Concerns and ways forward. *PLOS ONE*, 13(3):1–26.

[60] Marchi, E., Vesperini, F., Weninger, F., Eyben, F., Squartini, S., and Schuller, B. (2015). Non-linear prediction with lstm recurrent neural networks for acoustic novelty detection. In *Neural Networks (IJCNN), 2015 International Joint Conference on*, pages 1–7. IEEE.

[61] Mason, J. (2018). Cyber security statistics.

[62] McCallum, A., Nigam, K., and Ungar, L. H. (2000). Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178. ACM.

[63] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[64] Mockus, J. (2012). *Bayesian approach to global optimization: theory and applications*, volume 37. Springer Science & Business Media.

[65] Mockus, J., Tiesis, V., and Zilinskas, A. (1978). The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2.

[66] Münz, G., Li, S., and Carle, G. (2007). Traffic anomaly detection using k-means clustering. In *GI/ITG Workshop MMBnet*, pages 13–14.

[67] Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective. Adaptive Computation and Machine Learning*. MIT press.

[68] Nadaraya, E. A. (1964). On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142.

[69] Noor-Ul-Amin, M. (2010). Forecasting with neural networks: A comparative study using the data of emergency service. *arXiv preprint arXiv:1010.3501*.

[70] Olah, C. (2015). Understanding lstm networks.

[71] Pantic, M. (2014). Introduction to machine learning and case-based reasoning.

[72] Papalexakis, E. E., Beutel, A., and Steenkiste, P. (2014). Network anomaly detection using co-clustering. In *Encyclopedia of Social Network Analysis and Mining*, pages 1054–1068. Springer.

[73] Powers, D. and , A. (2011). Evaluation: From precision, recall and f-measure to roc, informedness, markedness and correlation. *J. Mach. Learn. Technol*, 2:2229–3981.

[74] Qi, Y. and Ishak, S. (2014). A hidden markov model for short term prediction of traffic conditions on freeways. *Transportation Research Part C: Emerging Technologies*, 43:95 – 111. Special Issue on Short-term Traffic Flow Forecasting.

[75] Rabiner, L. R. and Juang, B.-H. (1986). An introduction to hidden markov models. *ieee assp magazine*, 3(1):4 – 16.

[76] Rasmussen, C. E. and Williams, C. K. (2006). *Gaussian process for machine learning*. MIT press.

[77] Riesbeck, C. K. and Schank, R. C. (1989). *Inside Case-Based Reasoning*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.

[78] Sayad, S. (2019). Support vector machine - regression (svr).

[79] Sharda, R. and B. Patil, R. (1992). Connectionist approach to time series prediction: An empirical test. *Journal of Intelligent Manufacturing*, 3:317–323.

[80] Shyu, M.-L., Chen, S.-C., Sarinnapakorn, K., and Chang, L. (2003). A novel anomaly detection scheme based on principal component classifier. Technical report, MIAMI UNIV CORAL GABLES FL DEPT OF ELECTRICAL AND COMPUTER ENGINEERING.

[81] Smola, A. J. and Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222.

[82] Soni, D. (2018). Introduction to bayesian networks.

[83] Specht, D. F. (1991). A general regression neural network. *IEEE transactions on neural networks*, 2(6):568–576.

[84] Strang, G. (2016). *Introduction to Linear Algebra, 5th Edition*. Wellesley-Cambridge Press.

[85] Tapinos, A. and Mendes, P. (2013). A method for comparing multivariate time series with different dimensions. *PLOS ONE*, 8(2):1–11.

[86] Voyant, C., Notton, G., Kalogirou, S., Nivet, M.-L., Paoli, C., Motte, F., and Fouilloy, A. (2017). Machine learning methods for solar radiation forecasting: A review. *Renewable Energy*, 105:569–582.

[87] Warrender, C., Forrest, S., and Pearlmutter, B. (1999). Detecting intrusions using system calls: Alternative data models. In *Proceedings of the 1999 IEEE symposium on security and privacy (Cat. No. 99CB36344)*, pages 133–145. IEEE.

[88] Watson, G. S. (1964). Smooth regression analysis. *Sankhyā: The Indian Journal of Statistics, Series A*, pages 359–372.

[89] Watson, I. and Marir, F. (1994). Case-based reasoning: A review. *The Knowledge Engineering Review*, 9(4):327–354.

[90] Ye, N. and Chen, Q. (2001). An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International*, 17(2):105–112.

[91] Zhang, H., Zhang, W., Palazoglu, A., and Sun, W. (2012). Prediction of ozone levels using a hidden markov model (hmm) with gamma distribution. *Atmospheric Environment*, 62:64 – 73.

Appendices

Appendix A

UTS CBR Utility Libraries

Table A.1: TSFRESH feature calculators.

Feature	Meaning
abs_energy(x)	Returns the absolute energy of the time series which is the sum over the squared values
absolute_sum_of_changes (x)	Returns the sum over the absolute value of consecutive changes in the series x
agg_autocorrelation(x, param)	Calculates the value of an aggregation function f_{agg}
agg_linear_trend(x, param)	Calculates a linear least-squares regression for values of the time series that were aggregated over chunks versus the sequence from 0 up to the number of chunks minus one.
approximate_entropy(x, m, r)	Implements a vectorized Approximate entropy algorithm.
ar_coefficient(x, param)	This feature calculator fits the unconditional maximum likelihood of an autoregressive AR(k) process.
augmented_dickey_fuller (x, param)	The Augmented Dickey-Fuller test is a hypothesis test which checks whether a unit root is present in a time series sample.
autocorrelation(x, lag)	Calculates the autocorrelation of the specified lag
binned_entropy(x, max_bins)	First bins the values of x into max_bins equidistant bins.
c3(x, lag)	This function calculates the value of
change_quantiles(x, ql, qh, isabs, f_agg)	First fixes a corridor given by the quantiles ql and qh of the distribution of x.
cid_ce(x, normalize)	This function calculator is an estimate for a time series complexity [1] (A more complex time series has more peaks, valleys etc.).
count_above_mean(x)	Returns the number of values in x that are higher than the mean of x
count_below_mean(x)	Returns the number of values in x that are lower than the mean of x
cwt_coefficients(x, param)	Calculates a Continuous wavelet transform for the Ricker wavelet, also known as the "Mexican hat wavelet" which is

Table A.1 continued from previous page

Feature	**Meaning**
energy_ratio_by_chunks (x, param)	Calculates the sum of squares of chunk i out of N chunks expressed as a ratio with the sum of squares over the whole series.
fft_aggregated(x, param)	Returns the spectral centroid (mean), variance, skew, and kurtosis of the absolute fourier transform spectrum.
fft_coefficient(x, param)	Calculates the fourier coefficients of the one-dimensional discrete Fourier Transform for real input by fast
first_location_of_ maximum(x)	Returns the first location of the maximum value of x.
first_location_of_ minimum(x)	Returns the first location of the minimal value of x.
friedrich_coefficients(x, param)	Coefficients of polynomial h(x), which has been fitted to
has_duplicate(x)	Checks if any value in x occurs more than once
has_duplicate_max(x)	Checks if the maximum value of x is observed more than once
has_duplicate_min(x)	Checks if the minimal value of x is observed more than once
index_mass_quantile(x, param)	Those apply features calculate the relative index i where q% of the mass of the time series x lie left of i.
kurtosis(x)	Returns the kurtosis of x (calculated with the adjusted Fisher-Pearson standardized moment coefficient G2).
large_standard_deviation (x, r)	Boolean variable denoting if the standard dev of x is higher than 'r' times the range = difference between max and min of x.
last_location_of_ maximum(x)	Returns the relative last location of the maximum value of x.
last_location_of_ minimum(x)	Returns the last location of the minimal value of x.
length(x)	Returns the length of x
linear_trend(x, param)	Calculate a linear least-squares regression for the values of the time series versus the sequence from 0 to length of the time series minus one.
linear_trend_timewise(x, param)	Calculate a linear least-squares regression for the values of the time series versus the sequence from 0 to length of the time series minus one.
longest_strike_above_ mean(x)	Returns the length of the longest consecutive subsequence in x that is bigger than the mean of x
longest_strike_below_ mean(x)	Returns the length of the longest consecutive subsequence in x that is smaller than the mean of x
max_langevin_ fixed_point(x, r, m)	Largest fixed point of dynamics :math:argmax_x {h(x)=0}' estimated from polynomial h(x),
maximum(x)	Calculates the highest value of the time series x.
mean(x)	Returns the mean of x
mean_abs_change(x)	Returns the mean over the absolute differences between subsequent time series values which is
mean_change(x)	Returns the mean over the differences between subsequent time series values which is

Table A.1 continued from previous page

Feature	**Meaning**
mean_second_derivative _central(x)	Returns the mean value of a central approximation of the second derivative
median(x)	Returns the median of x
minimum(x)	Calculates the lowest value of the time series x.
v**number_crossing_m(x, m)**	Calculates the number of crossings of x on m.
number_cwt_peaks(x, n)	This feature calculator searches for different peaks in x.
number_peaks(x, n)	Calculates the number of peaks of at least support n in the time series x.
partial_autocorrelation(x, param)	Calculates the value of the partial autocorrelation function at the given lag.
percentage_of_reoccurrin _datapoints _to_all_datapoints(x)	Returns the percentage of unique values, that are present in the time series more than once.
percentage_of_reoccurring _val- ues_to_all_values(x)	Returns the ratio of unique values, that are present in the time series more than once.
quantile(x, q)	Calculates the q quantile of x.
range_count(x, min, max)	Count observed values within the interval [min, max).
ratio_beyond_r_sigma(x, r)	Ratio of values that are more than r*std(x) (so r sigma) away from the mean of x.
ratio_value_number_to _time_series_length(x)	Returns a factor which is 1 if all values in the time series occur only once, and below one if this is not the case.
sample_entropy(x)	Calculate and return sample entropy of x.
set_property(key, value)	This method returns a decorator that sets the property key of the function to value
skewness(x)	Returns the sample skewness of x (calculated with the adjusted Fisher-Pearson standardized moment coefficient G1).
spkt_welch_density(x, param)	This feature calculator estimates the cross power spectral density of the time series x at different frequencies.
standard_deviation(x)	Returns the standard deviation of x
sum_of_reoccurring _data_points(x)	Returns the sum of all data points, that are present in the time series more than once.
sum_of_reoccurring _values(x)	Returns the sum of all values, that are present in the time series more than once.
sum_values(x)	Calculates the sum over the time series values
symmetry_looking(x, param)	Boolean variable denoting if the distribution of x looks symmetric.
time_reversal_asymmetry _statistic(x, lag)	This function calculates the value of
value_count(x, value)	Count occurrences of value in time series x.
variance(x)	Returns the variance of x
variance_larger_than _standard_deviation(x)	Boolean variable denoting if the variance of x is greater than its standard deviation.