

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Streaming Framework for Adaptive Data Analytics in Industry 4.0

Diogo Alexandre Ferreira Granja e Silva



Mestrado em Engenharia Informática e Computação

Supervisor: Gil Manuel Magalhães de Andrade Gonçalves

Co-Supervisor: João Pedro Correia dos Reis

July 22, 2022

Streaming Framework for Adaptive Data Analytics in Industry 4.0

Diogo Alexandre Ferreira Granja e Silva

Mestrado em Engenharia Informática e Computação

July 22, 2022

Resumo

A Indústria 4.0 tem vindo a revolucionar os sistemas de produção como nunca antes visto. À medida que estes processos industriais são melhorados para fazer uso dos avanços tecnológicos, o mesmo acontece com as grandes quantidades de dados que estes produzem. Esses mesmos dados brutos permitem calcular diversas métricas de desempenho e outros indicadores que podem ser utilizados para otimizar esses mesmos sistemas e torná-los significativamente mais eficientes. Estas métricas geralmente dependem de modelos tradicionais de *Machine Learning* (ML) que pressupõem que os dados fornecidos por sensores, eventos-máquina e históricos de produção permanecerão estáticos.

No entanto, o mundo está em constante evolução – é dinâmico e não estacionário. Portanto, é inevitável que o desempenho destes modelos se comece a degradar ao longo do tempo à medida que a distribuição de dados e, portanto, a “*ground truth*” se altera. Isso representa um risco substancial para a indústria, já que pode significar atrasos significativos nas cadeias de produção. Por esta razão, tornou-se de extrema importância conseguir gerir o ciclo de vida dos modelos de ML de forma a estes se adaptarem aos desvios que ocorrem.

Para superar as falhas inerentes aos modelos tradicionais de ML, estes podem ser retreinados manualmente ou automaticamente, mediante algum gatilho, quando são detetadas anomalias nos dados que indiciam a ocorrência de desvios. Estas soluções mesmo assim não são perfeitas. Os algoritmos de ML e *streaming* adaptativos resolvem estes obstáculos efetuando retreino constante do modelo, dado um fluxo contínuo de dados. Isto traduz-se em modelos que se atualizam incrementalmente em tempo real e se adaptam aos desvios nos dados (*concept drift*). O desempenho do sistema é, portanto, melhorado ao fornecer previsões precisas com tempo de execução mínimo.

O trabalho desenvolvido para esta dissertação visa fornecer uma *framework* de processamento de *streaming* capaz de gerir o ciclo de vida dos algoritmos de ML tradicionais, monitorizando o desempenho do modelo e permitindo que seja retreinado quando necessário. Além disso, deve ser capaz de detetar desvio nos dados de algumas variáveis-chave, como anomalias nos sensores ou degradação da qualidade de fabrico. Por fim, providencia previsões adaptativas do comportamento de alguns indicadores como, por exemplo, conjetar a evolução da qualidade de um determinado produto ao longo do tempo.

A *framework* e os seus respetivos módulos são desenvolvidos em Python e implementados num ambiente de produção através de uma aplicação *web* desenvolvida com Streamlit e a sua própria API em Flask. Esta pilha de ML pretende fornecer uma ferramenta para testar o comportamento dos modelos em condições reais com diferentes configurações e abordagens.

Palavras-chave: concept drift, algoritmos de streaming, aprendizagem adaptativa, aprendizagem incremental, aprendizagem online, indústria 4.0

Abstract

Industry 4.0 has been revolutionizing production systems like never before. As these industrial processes are improved to make use of technological breakthroughs, so do the large amounts of data they consequently produce. That same raw data allows the computation of several performance metrics and other indicators that can be used to optimize such systems and make them significantly more efficient. These metrics usually depend on traditional Machine Learning (ML) models that assume data provided by sensors readings, machine events, and production logs will remain static.

Nevertheless, the world is constantly evolving – it is dynamic and non-stationary. Therefore, it is inevitable for model performance to start degrading over time as the data distribution, and thus, the “ground truth”, drifts. This poses a substantial risk for the industry as it can result in significant production delays. For this reason, it has become of utmost importance to be able to control the ML models’ life cycle and adapt to occurring drifts.

To overcome these inherent flaws of traditional ML models, the aforementioned models can be rebuilt manually or automatically, upon some trigger, when anomalies in the data indicate that drifting has happened. These solutions are still not perfect. Adaptive ML and streaming algorithms tackle the aforesaid issues by providing constant model retraining given a data stream – a continuous influx of data. This translates to models updating themselves incrementally in real-time and adapting to concept drift. The system’s performance is thus improved by delivering accurate predictions with minimal execution time.

The work completed for this dissertation provides a streaming processing framework capable of managing the traditional ML algorithms’ life-cycle by monitoring the model’s performance, allowing it to be retrained when necessary. Moreover, it provides the capability of detecting data drift of some key feature variables, derived from sensor anomalies or manufacturing quality degradation. Finally, it also delivers adaptive predictions of the indicators’ behavior, for instance, to forecast how a given product quality will evolve.

The framework and its respective modules are developed in Python and deployed to a production environment as a web application built on Streamlit with its own Flask API. This ML stack intends to provide a tool to test the models’ behaviors under real-world conditions with different settings and approaches.

Keywords: concept drift, streaming algorithms, adaptive learning, incremental learning, online learning, industry 4.0

Acknowledgements

I would like to start by thanking my supervisors, Prof. Gil Gonçalves and Prof. João Reis, for giving me the opportunity to work on this project at the DIGI2 lab. To Eliseu Pereira, also from the DIGI2 lab, a special thanks for his dedication and availability to guide me throughout this project.

I would also like to thank my family for always being there when I needed.

To my closest friends, thank you for living this adventure with me and making these five years unforgettable and definitely more enjoyable.

Finally, I wish to extend my thanks to everyone else not mentioned that, one way or another, contributed to this incredible journey.

Diogo Alexandre Silva

*“Data are becoming the
new raw material of business.”*

Craig Mundie

Contents

1	Introduction	1
1.1	Context and Motivation	1
1.2	Motivation	2
1.3	Objectives	2
1.4	Summary	2
2	State of the Art	4
2.1	Anomaly Detection	4
2.2	Time Series Forecasting	5
2.3	Machine Learning for Streaming Data	7
2.3.1	Anomaly Detection	7
2.3.2	Concept Drift Detection	8
2.3.3	Ensemble Learning	9
2.4	Related Work	10
3	Implementation	12
3.1	Problem Definition	12
3.2	Architecture	13
3.3	Technologies	14
3.4	Datasets	15
3.4.1	Anomaly Detection	15
3.4.2	Predictive Model	16
3.5	Exploratory Analysis	18
3.5.1	Data Cleaning	18
3.5.2	Correlation Analysis	19
3.5.3	Time Series Analysis for Forecasting	20
3.6	Algorithms	26
3.6.1	Batch Learning	26
3.6.2	Online Learning	27
3.7	Deployment	31
3.7.1	User Interface	32
3.7.2	Gateway Simulator	32
3.7.3	API	32
4	Experiments and Results	35
4.1	Experiments	35
4.1.1	Anomaly Detection	35
4.1.2	Time Series Forecasting	38

4.2 Framework	41
5 Conclusions and Future Work	45
5.1 Conclusions	45
5.2 Future Work	46
References	48

List of Figures

2.1	Different types of anomalies	5
2.2	Example of a 90-day forecast based on previous historical data	6
2.3	Different types of concept drift	9
3.1	Framework’s architecture	13
3.2	Temperature sensor data of an internal component of a large industrial machine (NAB)	15
3.3	Labeled anomalies of the temperature sensor data (NAB)	16
3.4	Correlation matrix	21
3.5	Autocorrelation of the time series for all time lags (hourly dataset)	22
3.6	Rolling statistics with a 30-day sliding window	22
3.7	Plot of the target value (% Silica Concentrate)	23
3.8	Plot with daily rolling statistics (moving average and Bollinger bands) for the target variable	24
3.9	Plot with 7-day rolling statistics for the target variable	24
3.10	Plot with 30-day rolling statistics for the target variable	24
3.11	Histogram and box plot of the target variable’s distribution of both the hourly and daily datasets	25
3.12	Time series decomposition: original data, trend, seasonality, and residuals	26
3.13	Framework’s user interface settings	33
4.1	Plot of the Flotation Column 01 Air Flow feature	36
4.2	Plot of the hourly average for the Flotation Column 01 Air Flow feature	36
4.3	Detected anomalies for the Flotation Column 01 Air Flow feature using the Half-Space Tree Detector with a constant threshold for the online learning approach	37
4.4	Detected anomalies for the Flotation Column 01 Air Flow feature using the One-Class SVM Detector from the <code>scikit-learn</code> library for the batch learning approach	37
4.5	Correlation matrix between the detected anomalies for the Flotation Column 01 Air Flow feature and the target variable (% Silica Concentrate)	38
4.6	Detailed plots of the ACF and PACF for the first 60 lags	38
4.7	Plot of the target variable’s (% Silica Concentrate) training and test data for the ARIMAS’s forecasting	39
4.8	Plot of the 24-hour interval to be forecasted by the ARIMA	39
4.9	Comparison of the original data with the ARIMA’s forecast results	40
4.10	Comparison of the original data with the auto-ARIMA’s forecast results	40
4.11	Batch learning training	41
4.12	MAE for the first few observations after training the batch regressor with 94 samples	41

4.13	MAE after training the batch regressor with 555 samples	42
4.14	MAE of the first few observations for the online regressor	42
4.15	MAE hikes after some more observations for the online regressor	43
4.16	Evolution of the Flotation Column 01 Air Flow feature throughout time (chosen for anomaly detection)	43
4.17	Plot and feedback message of the target correlation while no anomalies have been detected	44
4.18	Plot of the correlation between the detected anomalies and the target variable . .	44

List of Tables

2.1	Performance comparison using average AUC scores for different methods on benchmark datasets	8
3.1	Original mining dataset	18
3.2	Dataset's observations per hour and respective count	18
3.3	Incomplete hour blocks and number of samples missing	19
3.4	Resampled dataset without the incomplete hour blocks	19
3.5	Down-sampled dataset containing hourly observations	20
3.6	Down-sampled dataset containing daily observations	20
3.7	Results of the Augmented Dickey-Fuller (ADF) test	23
3.8	Results of the feature selection using the <code>SelectKBest</code> module	30
4.1	Anomaly detection results	36

Abbreviations

ADWIN	ADaptive WINdowing
ACF	AutoCorrelation Function
ADF	Augmented Dickey-Fuller
AIC	Akaike Information Criterion
API	Application Programming Interface
AR	Autoregression
ARF	Adaptive Random Forest
ARIMA	Autoregressive Integrated Moving Average
ARMA	Autoregressive Moving Average
AUC	Area Under receiver operating characteristic Curve
BoostHT	Online Coordinate Boosting
DDM	Drift Detection Method
DWM	Dynamic Weighted Ensemble
EDDM	Early Drift Detection Method
HST	Streaming Half-Space Trees
HT	Hoeffding Trees
HTTP	HyperText Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
LSTM	Long Short-Term Memory
MA	Moving Average
MAE	Mean Absolute Error
ML	Machine Learning
MSE	Mean Squared Error
NAB	Numenta Anomaly Benchmark
PAC	Probably Approximately Correct
PACF	Partial AutoCorrelation Function
RMSE	Root Mean Square Error
RNN	Recurrent Neural Network
RProp	Resilient backPropagation
RS-Trees	Randomized Space Trees
SARIMA	Seasonal Autoregressive Integrated Moving-Average
SGD	Stochastic Gradient Descent
SMAPE	Symmetric Mean Absolute Percentage Error
SRP	Streaming Random Subspaces
SUN	Semi-supervised classification algorithm for data streams with concept drifts and UNlabeled data

Chapter 1

Introduction

In recent years, industrial processes have been significantly improving, employing newly developed technologies that are increasingly becoming part of modern manufacturing systems. This chapter contains a brief contextualization (Section 1.1) and motivation (Section 1.2) of the dissertation, as well as its goals in Section 1.3. A summary of the document’s structure is also presented in Section 1.4.

1.1 Context and Motivation

Technological progress has been revolutionizing all fields of our society, including several industrial sectors. The Fourth Industrial Revolution, also known as Industry 4.0, is the name given to the innovative processes that are moving forward production systems and business models. This era of modernization is being driven by the aggregation of different disruptive technologies such as:

- **Big Data:** with the exponential increase of data generated and the ever-growing capacity to store and analyze such large amounts of information in real-time, decision-making processes become easier and more precise;
- **Data Analytics:** to make use of the growing amounts of data, it is necessary to employ techniques that analyze that information and transform it into useful insights that can be used to optimize processes or improve decision-making;
- **Machine Learning (ML):** this is the subfield of artificial intelligence that uses past data as input in order to predict new output values. It “learns” by leveraging Big Data to uncover key insights that ultimately drive decision-making processes;
- **Internet of Things (IoT):** machines, production systems, and equipment can be connected and interact with each other to form an integrated network. By having access to each other’s

sensors and data, the network can be intelligently managed to improve the system's efficiency.

1.2 Motivation

As Industry 4.0 revolutionizes industrial processes, larger and larger amounts of data are consequently produced [48]. Data from these sources have started to arrive in increasing volumes, with more velocity and greater variety – the three V's of Big Data.

“Industrial manufacturers are leveraging big data to transform their processes, their organizations and, in some cases, entire industries.” [26, p. 1]

This ever-growing amount of raw data motivated the computation of several performance metrics and other indicators with the intention of optimizing such systems and making them considerably more efficient. These metrics usually depend on traditional ML models, i.e., batch learning, that rely solely on data provided by sensor readings, machine events, and production logs that is assumed to be static.

However, the world is constantly evolving, and it is certain that a model's performance will diminish gradually as the data distribution drifts. Hence, batch learning models can quickly become obsolete and deliver subpar results. This poses a substantial risk for the industry because it can cause significant production delays.

1.3 Objectives

The work completed for this dissertation aims to provide a streaming processing framework capable of managing the traditional ML algorithms' life cycle by monitoring the model's performance, allowing it to be retrained when necessary. Moreover, it should be capable of detecting data drift of some key feature variables, derived from sensor anomalies or manufacturing quality degradation. Finally, it should also deliver adaptive predictions of the indicators' behavior, for instance, to forecast how a given product quality will evolve.

The framework and its respective modules will be developed in Python and deployed to a production environment as a web application built on Streamlit with its own Flask API. This ML stack intends to provide a tool to test the models' behaviors under real-world conditions with different settings and approaches.

1.4 Summary

In summary, this dissertation is organized as follows. Chapter 2 addresses the background, related work, and existing approaches to the main topics that surround the development of the streaming framework. In Chapter 3, the problem is disclosed, and a solution is proposed along with its architecture, used technologies, validation datasets and respective exploratory analysis, implemented

ML algorithms, and deployment. Chapter 4 demonstrates the results obtained with the different experiments, both individually and working together as a framework. Finally, Chapter 5 outlines the main takeaways regarding the work completed for this dissertation and also future work that could be developed on top of this project to improve it further.

Chapter 2

State of the Art

This literature review will be divided into three main topics of investigation: Anomaly Detection (Section 2.1), Time Series Forecasting (Section 2.2) and Machine Learning for Streaming Data (Section 2.3) – including Anomaly Detection for this specific time of learning, Concept Drift Detection and Ensemble Learning. Finally, Section 2.4 summarizes related work previously developed.

Some progress has been made in these areas, but taking a look at the industry’s current needs, there are still plenty of challenges that need to be addressed before any of these methods can efficiently replace existing methods in real-world scenarios.

This section intends to clarify the approaches that have been developed and the progress that has been made in both academia and industry, as well as the current limitations of that work.

2.1 Anomaly Detection

Industry 4.0 is a relatively new field that has been expanding rapidly. As such, it presents many interesting opportunities to explore the application of anomaly detection.

Anomaly Detection consists of the identification of unexpected observations within the data. It is highly specific to the individual use case, requiring expert knowledge of the method as well as the situation to which it is being applied [8].

Anomalies can be classified according to the way they occur. Point anomalies [21] can be described as occasional abnormal observations within a standard time series. Contextual anomalies [7] are observations that are inside the range of normal values for that time series, but deviate from the expected pattern at that specific timestamp. Finally, collective anomalies [7] comprise a collection of observations that can be considered abnormal only when they appear as a group. Figure 2.1 on page 5 depicts the three different types of anomalies.

For this dissertation, handled data will consist essentially of univariate time series data, that is, data output from a single source linked with the time of the observation [52].

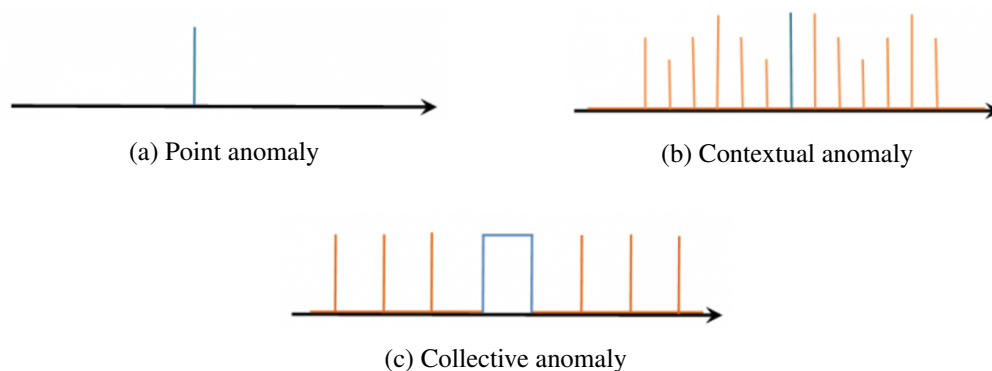


Figure 2.1: Different types of anomalies. The anomaly is colored in blue. [1]

To deal with this type of data, one can use a non-regressive approach. It is the simplest detection method and requires less computational power. Grubbs et al. [18] take into account historical data for that time series and compute its mean and variance. If the observation falls outside the range of this threshold, it is treated as an anomaly. The biggest limitation of this method is that it is unable to detect contextual and collective anomalies, as the temporal aspect of the time series is ignored.

Another popular and more sophisticated approach to this problem uses regression to identify outliers by applying some form of predictive time series modeling. The new observations are then compared with the predicted values, and the decision is made based on the difference between them. Bontemps et al. [4] present a Long Short-Term Memory (LSTM) network – a type of artificial Recurrent Neural Network (RNN) – that calculates an average error over time for a given time series. When this average surpasses a threshold, an anomaly is identified, allowing the system to detect collective anomalies more easily. The authors suggest that the main problem with this technique is that training data must be organized in a coherent manner to guarantee the stability of the system.

2.2 Time Series Forecasting

Some problems include a time component, usually as a form of a time series, requiring time series forecasting. Time series forecasting intends to predict a future value over a period of time based on previous data, since these series have an order dependence between observations, leading to the assumption that future trends will hold similar to historical trends [28]. An example can be seen on Figure 2.2 on page 6.

There are multiple models that can be used for time series forecasting, with varying results on the same data. Therefore, it becomes of utmost importance to determine the best model for that specific series, which can be achieved by performing a time series analysis. This technique comprises methods for extracting meaningful statistics and other characteristics of the data, like

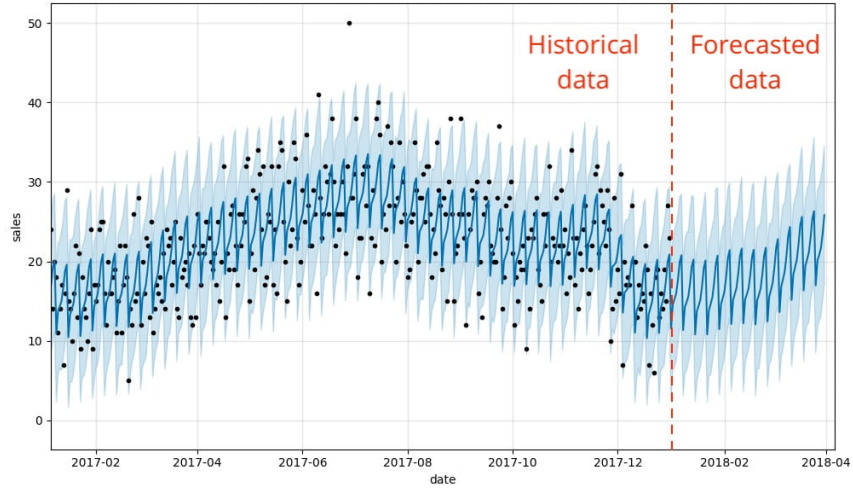


Figure 2.2: Example of a 90-day forecast based on previous historical data. [33]

seasonality and trends. Among the different time series forecasting methods, Decompositional models split a series into components:

- **Trend (T):** general direction of the data over time;
- **Seasonal (S):** behaviors that occur at specific regular intervals;
- **Residual (R):** everything not captured by the other components, i.e., “noise”.

A time series can also have a Cyclical (C) component that describes medium-term changes that repeat in cycles over a period of time. Nevertheless, the former three components can be combined either additively (Equation 2.1) or multiplicatively (Equation 2.2). Both trend and seasonality can be additive or multiplicative, resulting in 4 different combinations [38]. An additive trend suggests a linear trend, and an additive seasonality indicates the same frequency (width) and amplitude (height) of seasonal cycles [36]. A multiplicative trend indicates a non-linear trend (curved trend line), and a multiplicative seasonality indicates increasing/decreasing frequency (width) and/or amplitude (height) of seasonal cycles [36].

$$y_t = T_t + S_t + R_t \quad (2.1)$$

$$y_t = T_t \times S_t \times R_t \quad (2.2)$$

There are several types of time series methods used for forecasting, i.e., different ways to measure timed data, being the following the most common. Autoregression (AR), introduced by Slutsky [41], forecasts based on nearby previous values (lagged values). Moving Average (MA) – not to be confused with the smoothing technique with the same name – was suggested, more or less at the same time as AR, by Yule [51] and predicts values based on errors from previous predictions. Unlike AR, MA analyzes the errors in order to predict more correct values. Autoregressive Moving Average (ARMA) was first described by Whittle [45] and later popularized by Box et al. [5] by

combining AR and MA models to make predictions based on previous values and errors. Box et al. [5] also presented the Box–Jenkins method, also known as Autoregressive Integrated Moving Average (ARIMA), that, as the name suggests, takes into consideration the integration of the time series, i.e., the number of times it has to be differenced in order to become stationary. It is especially important since most time series data is non-stationary and needs to be differentiated. Finally, Seasonal Autoregressive Integrated Moving-Average (SARIMA) includes the seasonality contribution to the forecast, which ARIMA fails to take into account explicitly.

2.3 Machine Learning for Streaming Data

Data stream learning is a subfield of machine learning that updates its models given a continuous influx of data – potentially infinite – without performing multiple passes over data [17]. It is known by many other terms, such as online learning, incremental learning, or sequential learning.

Online models (usually) naturally adapt to drift, and their development is closer to reality. However, due to its relative novelty, it has not been yet thoroughly investigated, hence the lack of adoption in the industry.

Machine learning for streaming data is a complex process that differs a lot from the traditional model's approach. Even so, the prime distinction from batch learning, i.e., traditional ML models, is that learning is achieved one observation at a time. This translates into models not requiring to be retrained. Nevertheless, for the purposes of this literature review, the focus will be on the learning process.

Learning from a data stream is a continuous process where conditions constantly evolve. As such, there are many essential aspects to consider in this and changes to be monitored during the learning procedure [17].

2.3.1 Anomaly Detection

Data streams pose an unprecedented challenge to most existing anomaly detection techniques, as they need to be able to sustain the three V's of Big Data:

- **Volume:** conventional detection algorithms rely on storing observations in memory to build their models, which is not possible in the case of data streams due to the exceptional volume that they can uphold;
- **Velocity:** both the processing of the observations and the updating of the anomaly detection mechanism need to be just as fast as the inherent high speed with which streaming data arrives;
- **Variety:** because of the continuous nature of streaming data, drifting also evolves continuously with new concepts. This can happen when new changes might startle models that were still learning previous patterns. Therefore, the detector needs to be able to adjust to the evolution of what the normal behavior evolves to over time [47].

Finally, labeled anomalous data is usually rare or not available for training in streaming data, so detection systems must be capable of detecting suspicious behavior, having mostly only trained with normal data [47].

For example, Hoeffding Trees (HT) [11] – a decision tree induction algorithm – along with Online Coordinate Boosting (BoostHT) [37], are two well-studied proposals for anomaly detection in streaming data. However, both HT and BoostHT require labeled anomalies during training, which is not a realistic assumption for most systems.

Apart from supervised methods discussed earlier, more and more semi-supervised learning algorithms, i.e., methods trained with a small amount of labeled data and a large amount of unlabeled data, have emerged to solve anomaly detection problems involving streaming data. For instance, Streaming Half-Space Trees (HSTa) [7] is a semi-supervised one-class learner that performs regular updates of its model. Tan et al. [43], on the other hand, also uses Streaming HS-Trees, but without model update (named HSTn), learning only from the first few given instances of the stream. A more novel approach, proposed by Wu et al. [47], consists of a fast density estimator implemented by multiple fully randomized space trees (RS-Trees), named RS-Forest.

Both Tan et al. [43] and Wu et al. [47] used synthetic (MULCROSS [39]) and real-world (SMTP, HTTP, SMTP+HTTP [24] and COVERTYPE, SHUTTLE [12]) datasets to calculate the AUC (Area Under receiver operating characteristic Curve) [19] and compare the performances of the aforementioned anomaly detectors. The results were summarized in Table 2.1 on page 8.

Table 2.1: Performance comparison using average AUC scores for different methods on benchmark datasets. [47, 43]

Dataset	# Instances	Anomaly	AUC				
			HT	BoostHT	HSTa	HSTn	RS-Forest
HTTP	567 497	attack (0.4%)	.994	.998	.996	.982	.999
SMTP	95 156	attack (0.03%)	.858	.692	.875	.740	.880
SMTP+HTTP	662 653	attack (0.35%)	.991	.993	.996	.387	.998
CONVERTYPE	286 048	outlier (0.9%)	.998	.968	.991	.854	.995
MULCROSS	262 144	2 dense clusters (10%)	1.00	1.00	.998	.998	1.00
SHUTTLE	49 097	class 2,3,5–7 (7%)	.991	.984	.999	.999	.998

2.3.2 Concept Drift Detection

Concept drift can be characterized by unexpected changes over time to the statistical properties of the target variable that the model is trying to predict [30]. If models are not able to adapt to concept drift, learning performance will deteriorate over time.

There are two types of concept drift [14], illustrated in Figure 2.3 on page 9:

- **Real Drift:** data changes and causes a deterioration of the model's performance;
- **Virtual Drift:** the data's underlying distribution changes, but the model's performance was not affected.

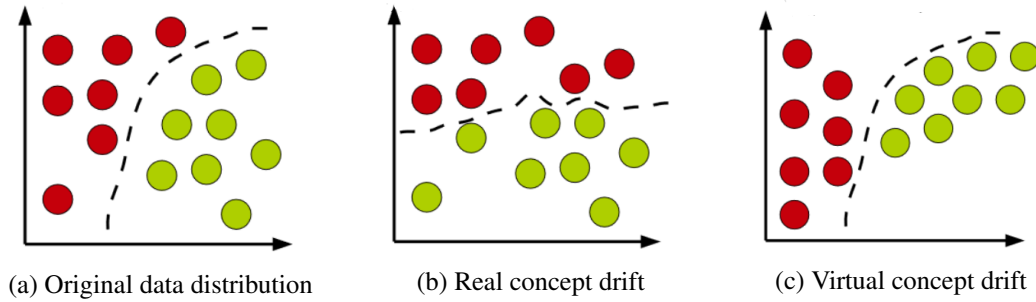


Figure 2.3: Different types of concept drift. [14]

In the case of drift detection, the objective is to detect when the data of a given distribution changes. For example, in the case of classifiers, a suitable detector is one that maximizes true positives and minimizes false positives whilst having a short detection delay. For this, detection algorithms receive from the learner as input a univariate stream of predictions.

Drift Detection Method (DDM) is a popular concept drift detector proposed by Gama et al. [14] based on the statistical process control. It relies on the premise of the probably approximately correct (PAC) [31] learning model, i.e., as long as the data distribution is stationary, if the number of analyzed samples increases, the learner's error rate will consequently decrease. If the error rate increases, surpassing a calculated threshold, the algorithm will detect it or warn the user that a change is imminent – the so-called warning zone.

Early Drift Detection Method (EDDM) [2] tries to enhance the detection rate of DDM, while remaining unaffected by sudden concept drift changes. Instead of keeping track of only the error rate, it registers running the average difference and standard deviation between two error values, as well as their maximums.

Apart from statistical methods, methods based on monitoring the data distributions of different time windows are also used. ADaptive WINdowing (ADWIN) [3] is a popular drift detection method with mathematical guarantees. ADWIN stores a variable-length window of recent items in order to assess if there has been a change in the data distribution. This is done by comparing the average of different sub-windows and evaluating if they correspond to the same distribution.

However, it is essential to highlight that, for all the aforementioned methods, the ability to detect drifts decreases sharply if the ground truth is not immediately available after the prediction is made [17].

An algorithm focusing on delayed or partially labeled streams is SUN (Semi-supervised classification algorithm for data streams with concept drifts and UNlabeled data) [27], a clustering algorithm that produces concept clusters at leaves on an incremental decision tree.

2.3.3 Ensemble Learning

Despite the promising results achieved by online learning, it is valuable to highlight that this type of learning is still recent and might underperform (when compared with the results of batch

learning approaches) in some situations. This is where ensemble learning – combining several algorithms to obtain better predictive performance – becomes useful. This technique is usually successful and can be used to improve anomaly and concept drift detection, as well as other problems out of the scope of this literature review, such as concept evolution and feature evolution.

Concept drift adaption can be classified into reactive – an approach that observes the drift points in the data streams, adapting the model only when triggered – and proactive, that continuously adapts the model after every observation [49].

An example of a reactive strategy to cope with concept drift using ensemble learning is Dynamic Weighted Ensemble (DWM) [49], which, as the name suggests, predicts using a weighted-majority vote of weak learners, converting them to strong ones – experts. It then dynamically creates and deletes them according to the evolution of their performance.

On the other hand, a proactive strategy is ADWIN Bagging. It is a popular strategy that combines ADWIN [3] and online bagging [35].

Strategies that combine reactive and proactive strategies have also been studied, being Adaptive Random Forest (ARF) [15] and Streaming Random Subspaces (SRP) [16] good examples. These are adaptations to streaming data of the Random Forest [6] and Random Patches algorithms [29], respectively, the main difference being that ARF is based on local subspace randomization and SRP uses a global subspace randomization strategy [17].

Khine et al. [25] presents a framework that takes in the benefits of assembling technology based on miss classification stream data. The authors clarify that their experiments were compared with popular ensemble techniques such as Boosting and Bagging and showed that accuracy rates increased, and classification time was reduced.

2.4 Related Work

Most of the traditional and existing solutions up until recently completely ignored the temporal aspects of the data and focused only on “offline” scenarios. This proves to be inefficient, especially when vast amounts of data are involved. As a result, the ever-increasing data sources have encouraged the study and development of frameworks capable of dealing with data streams. The goal of these solutions is to make use of the powerful paradigm of online learning for systematic analysis and identification of patterns in the observations.

Concerning specifically anomaly detection on streaming data, Yilmaz et al. [50] created an open-source framework in Python named PySAD. It provides several methods for streaming anomaly detection and a complete set of tools to design experiments.

Vidhyut et al. [44] employ the advantages of online learning-based models for crime analysis. According to the authors, crime naturally follows a geographic and demographic pattern, requiring a constant model update when dealing with such data, making it suitable for the application of these methods.

Another example of related work is presented in [10]. The authors propose a "distributed streaming framework for spatial-temporal data clustering". This project explores the utilization of

such framework on the massive data generated from sensors and other mobile equipment. These devices produce data in real-time that can be inspected by the framework to detect events and extract certain patterns that otherwise, i.e., using traditional machine learning models, would not be efficiently detected. The results proved effective and conveyed the effectiveness and efficiency of these novel techniques.

Chapter 3

Implementation

This chapter outlines the existing problem and explains how the proposed solution was used to tackle it.

The first section of this chapter (Section 3.1) formalizes the problem. Section 3.2 presents the architecture of the framework. Section 3.3 confers about technologies used to build this particular solution, while Section 3.4 describes the datasets that attested the feasibility of this work. Before laying out the batch and online model algorithms in Section 3.6, the exploratory analysis and preparation of the data is detailed in Section 3.5. At last, Section 3.7 elaborates on the deployment, including the respective API and User Interface, of the framework.

3.1 Problem Definition

Building a model requires learning from existing data and using that acquired knowledge to make predictions on new data whose answer is unknown. This can be seen as the problem of approximating a mapping function f given input data X to predict an output value y (Equation 3.1).

$$y = f(X) \tag{3.1}$$

This mapping is often assumed to be constant, suggesting that what was learned by the model will remain true indefinitely. Nevertheless, even though this can be true for many problems, the world is constantly evolving – it is dynamic and non-stationary. Therefore, it is inevitable for model performance to start degrading over time as the data distribution, and thus, the “ground truth”, conveyed by the underlying mapping function, drifts [32]. This poses a substantial risk for the industry as it can result in significant production delays.

For this reason, it has become of utmost importance to be able to control the ML models’ life cycle and adapt to occurring drifts.

To address the inherent flaws of batch learning, there are some different approaches with distinct outcomes. These consist of simple strategies that can evolve into more complex processes [32].

A somewhat naive but straightforward option is to periodically gather new and updated data and use it to batch retrain the model from scratch, manually. This can be improved by automatically triggering or scheduling model retraining when anomalies in the data indicate that drifting has happened.

However, these solutions are still not perfect. A model's predictive performance is expected to decline as soon as it is built, and the only way to fully tackle this issue is by utilizing adaptive ML and streaming algorithms to provide constant model retraining given a data stream – a continuous influx of data. This translates to models updating themselves incrementally in real-time and adapting to concept drift. The system's performance is thus improved by delivering accurate predictions with minimal execution time.

3.2 Architecture

To solve the aforementioned problem, a machine learning framework was designed and developed, whose architecture can be seen in Figure 3.1 on page 13.

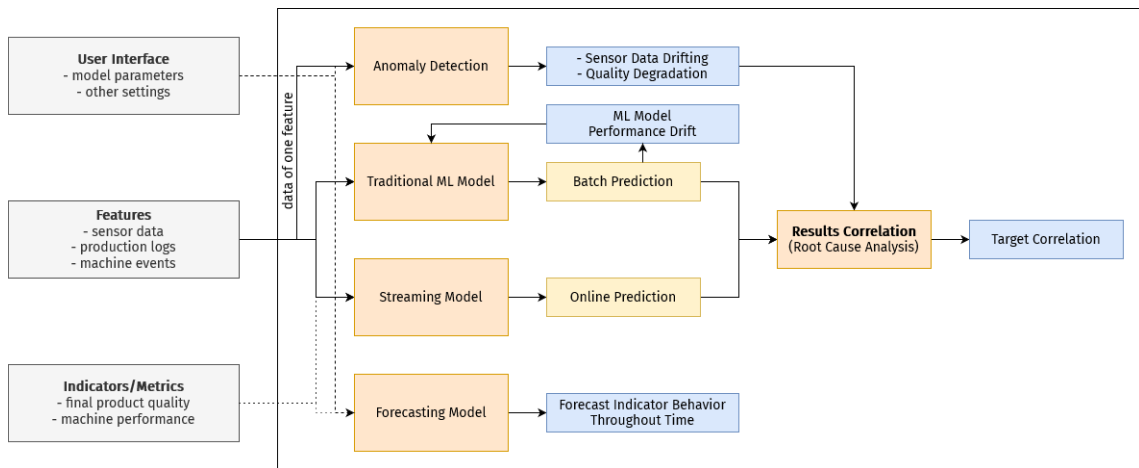


Figure 3.1: Framework's architecture.

The framework has the following three inputs:

- **Features:** data that comes, for example, from sensors, production logs, or machine events;
- **Metrics or Indicators:** target indicators, such as the final product's quality and other machine performance metrics;
- **User Interface:** model parameters and other settings that are used to change the behavior of the framework and its results.

The framework is composed of a traditional batch learning and a streaming online model module. The output of the traditional ML model is evaluated for possible performance drifting so that it can be retrained. The anomaly detection module is responsible for tracking concept drift of any given data feature, signaling possible sensor anomalies or manufacturing quality degradation. Both batch and online predictions and the detected anomalies are used to compute the target correlation in order to try to explain the variations of the results and perform a root cause analysis. Finally, a forecasting model aims to forecast some of the indicators' behavior throughout time, like the evolution of the final product quality.

3.3 Technologies

This framework was built using Python 3. The traditional batch learning approach was based on scikit-learn¹ – a Python module built on top of SciPy². This package is the industry's gold standard and provides simple and efficient tools for data analysis and machine learning approaches, including regression and clustering.

For the development of the online machine learning module, RiverML³ – the result of a merger between crème⁴ and scikit-multiflow⁵ libraries – was used. River is a library created specifically for online machine learning models that operate on data streams, regardless of where the data comes from, i.e., reactive or proactive streams:

- **Reactive:** reactive streams consist of data that cannot be controlled, as it is the result of an event, being the only possible outcome to “react” and handle it as intended;
- **Proactive:** proactive streams comprise data that can be controlled as they derive from static sources.

Although the traditional approach is to turn reactive streams into proactive datasets, the real challenge is to ensure proactive models that are easily trained offline perform well in reactive production environments. The main reason why traditional batch learning models – but also all machine learning models, offline or online – might underperform in production is because of concept drift. The advantage of online models over offline is that, contrary to traditional batch learning techniques that train a model on an entire data batch at a time, online models process each observation coming from a data stream one at a time. Therefore, online models are stateful and dynamic as they learn continuously, iteratively. This is why online machine learning shines in such environments, as the built models not only make predictions on the incoming streaming data but also learn from it on the fly, adapting seamlessly to drifting.

Unlike its traditional batch learning counterpart, where data is usually split into a training (learning) and testing (evaluation) dataset, online model evaluation implies learning and inferring

¹<https://github.com/scikit-learn/scikit-learn>

²<https://scipy.org/>

³<https://github.com/online-ml/river>

⁴<https://pypi.org/project/creme/>

⁵<https://github.com/scikit-multiflow/scikit-multiflow>

in the same order as what would happen in production, which ensures these models will perform just as well in production. This is especially important when dealing with time-series data that might be autocorrelated. Moreover, to emulate a production scenario as closely as possible while developing the models without a real-time feed, `River` allows offline datasets to be read in an online manner, one sample at a time, in order to ensure the model will perform just as well in production.

One final detail to bear in mind between these paradigms is that, in contrast to batch learning, in `River`, dictionaries are the basic building block as vectorization does not bring any speedup to this paradigm. This is due to the fact that online processing is entirely different from batch processing, and thus, numeric processing libraries such as `numpy`⁶ and `PyTorch`⁷ bring nothing but overhead, so it is way more efficient to use native Python data structures.

3.4 Datasets

3.4.1 Anomaly Detection

To benchmark and tune the anomaly detector on its own, Numenta Anomaly Benchmark (NAB)⁸ was used. This dataset was specially conceived to test and score the obtained results in a controlled environment. According to the authors, “it comprises over 50 labeled real-world and artificial time series data files”, but for this particular case, a dataset containing real data and a real known cause was used.

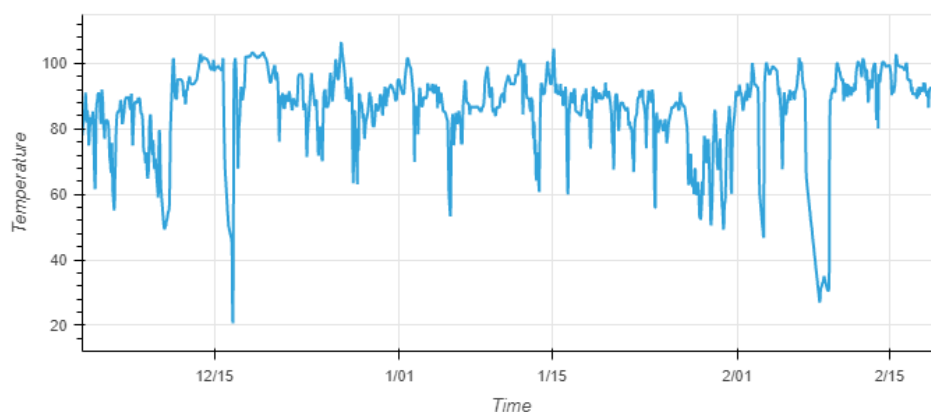


Figure 3.2: Temperature sensor data of an internal component of a large industrial machine, derived from NAB’s `machine_temperature_system_failure.csv` file.

This data contains three anomalies, as highlighted in Figure 3.3 on page 16: the first anomaly is a planned shutdown of the machine. The second is difficult to detect and directly led to the third anomaly, a catastrophic failure of the machine.

⁶<https://numpy.org/>

⁷<https://pytorch.org/>

⁸<https://www.kaggle.com/boltzmannbrain/nab>

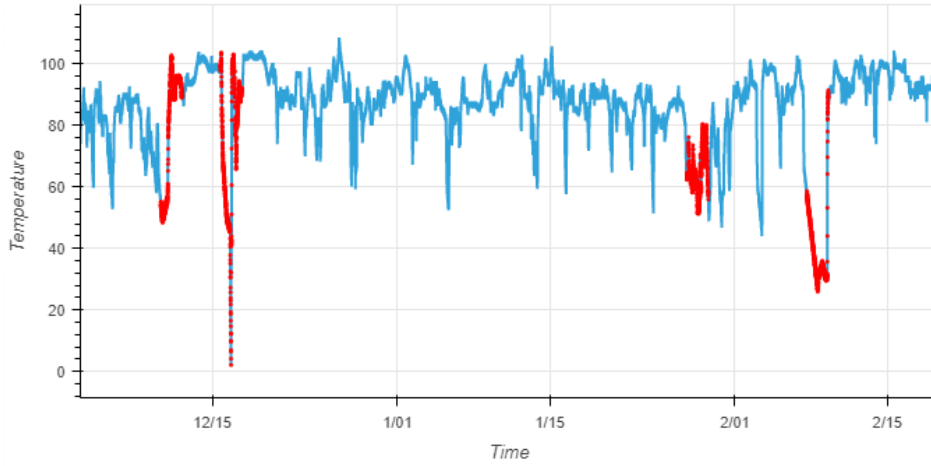


Figure 3.3: Labeled anomalies (in red) of the temperature sensor data.

By using this dataset, which contains labeled abnormal observations, it becomes possible to evaluate the quality of the detectors. This can be done by computing the F1-score, a metric that combines the precision and recall into a single metric by taking their harmonic mean (Equation 3.2).

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (3.2)$$

The detectors were then further tested with another dataset containing actual industry data. However, because such anomalies (just like in real industry data) are not labeled, the evaluation cannot be measured with metrics such as the F1-score.

3.4.2 Predictive Model

Finding reliable datasets from real-world manufacturing plants is a challenge itself. The plan was to feed the framework with data from a product manufacturing plant, but most of the reliable datasets were image datasets. For instance, the dataset Casting Product Image Data for Quality Inspection⁹ intends to be used on deep learning classification models to automate the inspection process. Nevertheless, by extending the search to other types of manufacturing facilities, it was possible to filter out a suitable dataset – Quality Prediction in Iron Ore Mining¹⁰. This dataset comes from a database of a real-world iron ore mine and gathers data from one of the most critical parts of a mining process – froth flotation – which comprises the process of selectively separating hydrophobic from hydrophilic materials [46].

With this data, the level of impurity in the ore concentrate, i.e., the percentage of silica, can be predicted, which is going to be the target variable, sampled on an hourly basis. This prediction is

⁹<https://www.kaggle.com/datasets/ravirajsinh45/real-life-industrial-dataset-of-casting-product>

¹⁰<https://www.kaggle.com/edumagalhaes/quality-prediction-in-a-mining-process>

important because, even though it can be measured, it results from a lab measurement, requiring at least a one-hour processing of the ore by process engineers.

The data ranges from March 2017 up until September 2017. Some of the metrics are sampled every 20 seconds, i.e., 180 times per minute, while others are sampled once an hour. Features are divided and described by the author as follows:

- **% Iron Feed:** percentage of Iron (purity) that comes from the iron ore that is being fed into the flotation cells;
- **% Silica Feed:** percentage of silica (impurity) that comes from the iron ore that is being fed into the flotation cells.

These two features measure the quality of the iron ore pulp as it comes to the flotation plant, right before it is fed to it.

- **Starch Flow:** starch (reagent) flow measured in m³/h;
- **Amina Flow:** amina (reagent) flow measured in m³/h;
- **Ore Pulp Flow:** flow measured in t/h;
- **Ore Pulp pH:** pH scale from 0 to 14;
- **Ore Pulp Density:** density scale from 1 to 3 kg/cm³.

These five features represent crucial variables that impact the quality of the final ore product.

- **Flotation Column 0{N} Air Flow:** air flow that goes into the flotation cell measured in Nm³/h with N ranging from 1 to 7;
- **Flotation Column 0{N} Level:** froth level in the flotation cell measured in mm (millimeters) with N ranging from 1 to 7.

These features (fourteen in total) showcase data that comes from the process itself and measures the airflow inside the flotation columns, which also impacts the final ore quality.

- **% Iron Concentrate:** percentage of iron is presented at the end of the flotation process (0-100%, lab measurement);
- **% Silica Concentrate:** percentage of silica is presented at the end of the flotation process (0-100%, lab measurement).

Finally, these two features depict the quality measurements from the labs. It is important to highlight that these two metrics are highly, and inversely, correlated, as a higher percentage of one translates to a lower percentage of the other, i.e., the total sum should be 100%.

3.5 Exploratory Analysis

As mentioned in the previous section (Section 3.4.2), an iron ore mining dataset was used to test the framework. The aim was to predict the percentage of silica (impurity) at the end of the mining process.

Table 3.1: Original mining dataset.

Date	% Iron Feed	% Silica Feed	...	% Silica Concentrate
2017-03-10 01:00:00	55,2	16,98	...	1,31
2017-03-10 01:00:00	55,2	16,98	...	1,31
...	...	...	...	...
2017-09-09 23:00:00	49,75	23,2	...	1,71

736282 rows × 24 columns

3.5.1 Data Cleaning

This is a considerably large dataset containing 736,282 observations and 24 feature columns, starting on the 10th of March 2017 at 01:00 until the 9th of September 2017 at 23:00. Before doing anything with the data, it is vital to realize that real-world data is messy and missing values can lead to biased or invalid results. Starting by checking for null (NA) values, it was possible to ascertain that the available dataset was not missing any values for the observations it contained. After some further analysis, it was possible to understand why there were not any null values earlier. It turned out that there were some rows, i.e., observations, themselves missing (Table 3.2 on page 18).

Table 3.2: Dataset's observations per hour and respective count.

Samples per Hour	Count
180	4080
1	4
179	2
175	1
174	1
170	1
166	1
164	1
143	1
141	1
134	1
99	1
96	1
58	1

As the observations whose number of samples per hour was less than the expected 20-second interval did not seem to have any meaningful explanation (Table 3.3 on page 19), it was best

to remove them altogether so that it was possible to work with a time series that could reliably simulate a steady stream of readings coming from an industrial setting.

Table 3.3: Incomplete hour blocks and number of samples missing.

Date (hour block)	Samples per Hour
2017-03-10 01:00:00	174
2017-03-12 01:00:00	179
2017-04-10 00:00:00	179
2017-06-01 15:00:00	175
2017-06-28 15:00:00	166
2017-06-28 16:00:00	96
2017-06-29 14:00:00	170
2017-06-29 15:00:00	164
2017-07-11 06:00:00	99
2017-07-11 07:00:00	1
2017-07-11 08:00:00	1
2017-07-11 09:00:00	1
2017-07-11 10:00:00	134
2017-07-13 14:00:00	58
2017-07-13 15:00:00	1
2017-07-13 16:00:00	141
2017-07-26 11:00:00	143

This operation did not compromise the quality of the remaining data, as it only removed a total of 1,882 data points, bringing the total down to 734,400 observations. This enabled the dataset to be resampled in equal 20-second intervals, as seen in Table 3.4 on page 19.

Table 3.4: Resampled dataset without the incomplete hour blocks.

Date	% Iron Feed	% Silica Feed	...	% Silica Concentrate
2017-03-10 02:00:20	55,2	16,98	...	1,11
2017-03-10 02:00:40	55,2	16,98	...	1,11
...	...	...	...	...
2017-08-27 02:00:00	49,75	23,2	...	1,71

734400 rows × 24 columns

To make testing easier, two additional datasets, considerably smaller than the original one, were created by down sampling the original dataset: one with 4,081 hourly observations (Table 3.5 on page 20) and another with 171 daily observations (Table 3.6 on page 20).

3.5.2 Correlation Analysis

With the data cleaned and prepared to be used, it was important to start with some Correlation Analysis of the features. This analysis can unveil meaningful relationships between features and uncover interesting insights and interdependencies between them. For this, Pearson's method was used. It is a method that not only returns the magnitude of the association, i.e., the correlation,

Table 3.5: Down-sampled dataset containing hourly observations.

Date	% Iron Feed	% Silica Feed	...	% Silica Concentrate
2017-03-10 02:00:00	55,2	16,98	...	1,11
2017-03-10 03:00:00	55,2	16,98	...	1,269111
...	...	...	...	...
2017-08-27 02:00:00	49,75	23,2	...	1,71

4081 rows $\times$ 22 columns

Table 3.6: Down-sampled dataset containing daily observations.

Date	% Iron Feed	% Silica Feed	...	% Silica Concentrate
2017-03-10	55,384847	17,217287	...	1,763791
2017-03-11	59,842484	9,176868	...	3,102428
...	...	...	...	...
2017-08-27	49,75	23,2	...	1,834903

171 rows $\times$ 22 columns

but also the direction of the relationship. Correlation coefficients range between +1 and -1, with the extremes (± 1) meaning a total association between the two features. The closer the value is to 0, the weaker this relationship is. A positive coefficient indicates a positive relationship, while a negative coefficient indicates a negative relationship.

The matrix on Table 3.4 on page 21 shows the correlations between all the features. From it, the features which affect the % Silica Concentrate target the most can be identified. There are some key conclusions that can be drawn from a quick analysis of this matrix. First, the percentage of Iron Concentrate is highly correlated to the percentage of Silica Concentrate obtained from the lab readings, which confirms the initial supposition. Second, this premise also goes for the relation between the percentage of Iron Feed and percentage of Silica Feed (as they come to the flotation plant), i.e., they are also highly correlated as the higher the presence of one translates into a lower presence of the other. This renders these features redundant, since they convey the same information.

3.5.3 Time Series Analysis for Forecasting

As the name suggests, a time series is a series of data points ordered chronologically. Before designing and testing models for forecasting, it is essential to analyze the series they will work on in order to identify underlying phenomenons, trends, cyclicity, and seasonality.

- **Autocorrelation Analysis:** determine whether observations are related to each other throughout different points in time;
- **Stationarity Analysis:** find out if the time series is stationary, as most models, including ARIMA, required them to be;
- **Trend Analysis:** determine whether if there is an underlying trend;

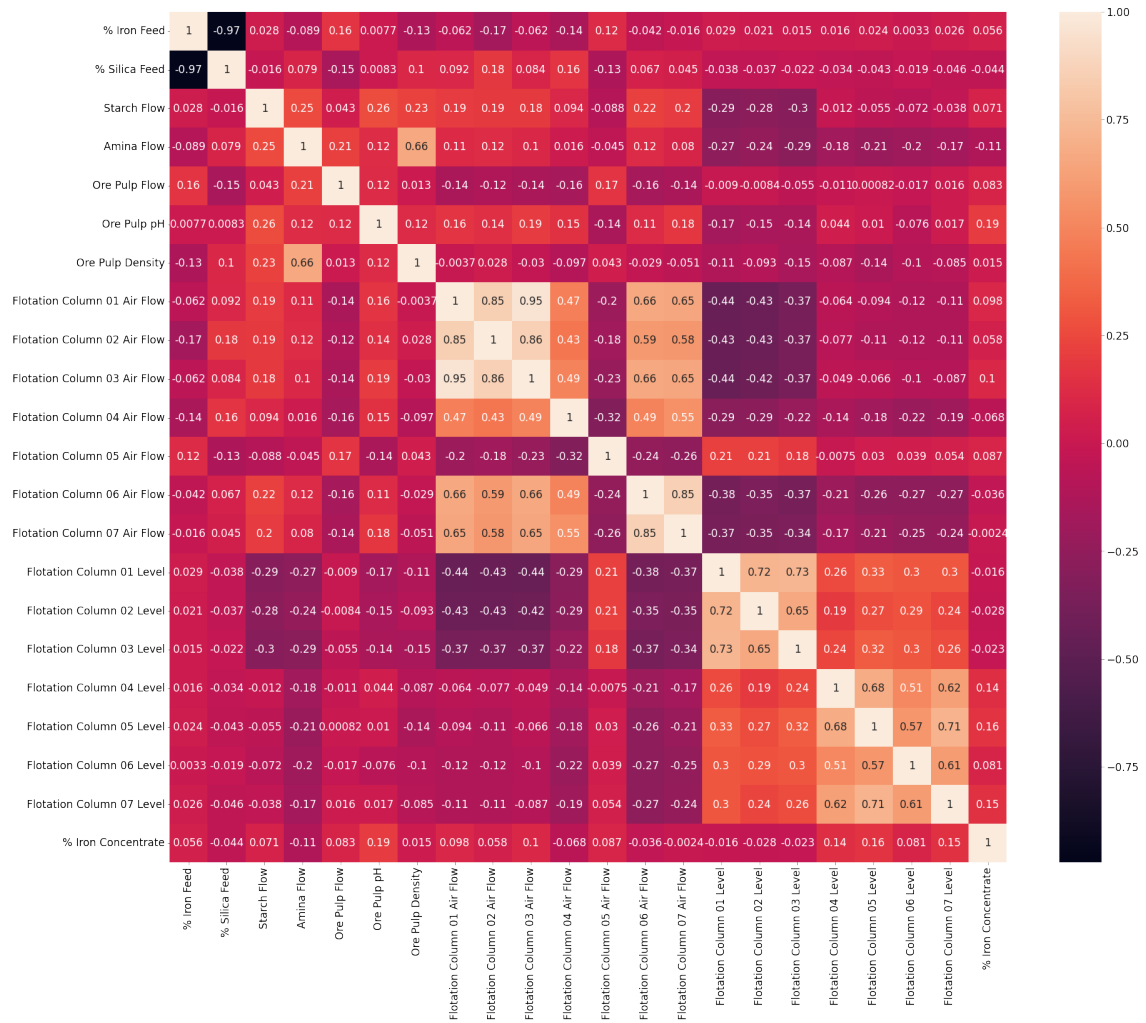


Figure 3.4: Correlation matrix.

- **Outliers Detection:** understand if some data points differs significantly from other observations;
- **Seasonality Analysis:** determine the best seasonal parameter (s) for this series.

3.5.3.1 Autocorrelation Analysis

The fact that a time series data is ordered makes it often have serial dependence, i.e., autocorrelation. Autocorrelation refers to the degree of similarity between A) a given time series and B) a lagged version of itself, over C) successive time intervals [23]. In other words, autocorrelation measures the relationship between a variable's present value and any of its past values.

The Auto Correlation Function (ACF) takes all the observations and calculates the correlation (Pearson's) between them and their own lagged value without controlling other lags, i.e., regardless of their effect on the future or present time period. From Figure 3.5 on page 22, we see that ACF values are only within the 95% confidence interval (represented by the solid gray line) for lags

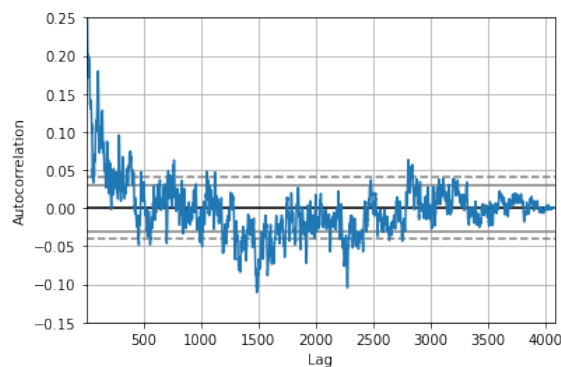


Figure 3.5: Autocorrelation of the time series for all time lags (hourly dataset).

greater than around 400, which confirms that this data is significantly autocorrelated. The values for the ACF also seem to indicate that the time series is stationary, as it quickly degrades to values around zero instead of slowly decreasing over time as it happens on non-stationary series.

3.5.3.2 Stationarity Analysis

Notwithstanding the initial indications given by the correlogram (Figure 3.5 on page 22), for a time series to be considered stationary, all statistical properties – mean, variance and covariance – need to remain constant over time. This analysis can be done visually or by performing a statistical test, namely the Dickey-Fuller Hypothesis [9] testing. The easiest and most logical way to start is by plotting rolling statistics (Figure 3.6 on page 22) because if the mean and variance change over time, then it means the series is non-stationary.

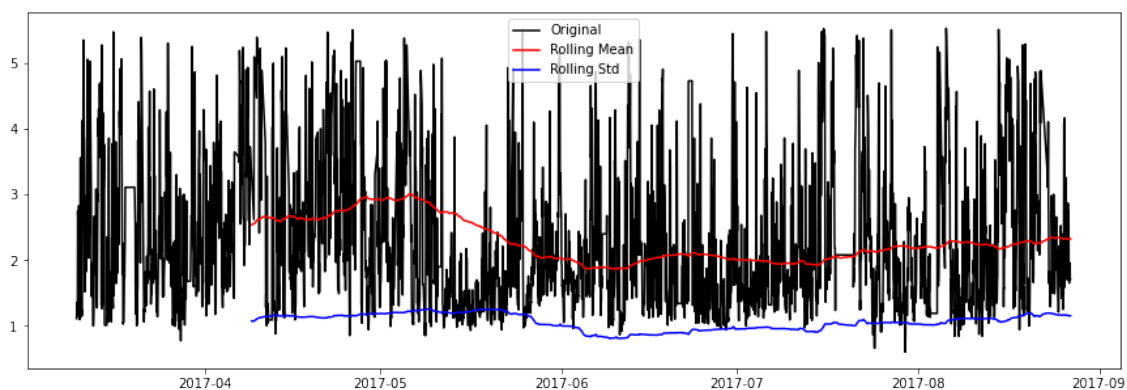


Figure 3.6: Rolling statistics with a 30-day sliding window.

In this case, the rolling statistics seem to indicate that the series is stationary. More formal approaches to identify stationarity include statistical tests. The Dickey-Fuller test is a unit root test that tests the null hypothesis of a series being non-stationary given the presence of unit root in an autoregressive model [9].

Table 3.7: Results of the Augmented Dickey-Fuller (ADF) test.

Item	Value
Test Statistic	-8.124231
p-value	1.134481×10^{-12}
#Lags Used	24
Number of Observations Used	4056
Critical Value (1%)	-3.431963
Critical Value (5%)	-2.862253
Critical Value (10%)	-2.567149

The results of the Augmented Dickey-Fuller (ADF) test, an extension of the original test, show that the test statistic (-8.124231) is less than the 1% critical value (-3.431963), indicating the time series is stationary with a 99% confidence. The p-value (1.134481×10^{-12}) is also way below the 0.05 threshold, rejecting the null hypothesis, and further confirming the previous indications.

3.5.3.3 Trend Analysis

The trend is the component of a time series that represents variations of low frequency in a time series, the high and medium frequency fluctuations having been filtered out [34]. The best way to determine if the data follows an underlining trend, be it linear or non-linear, is by plotting it [38].

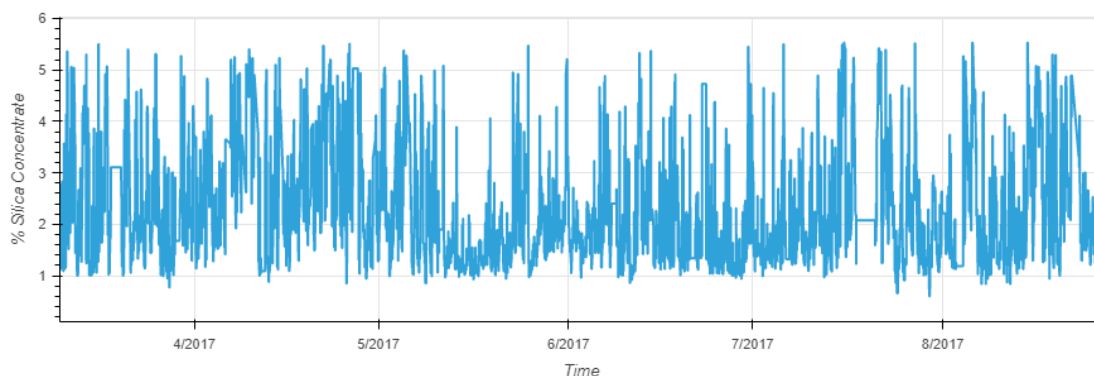


Figure 3.7: Plot of the target value (% Silica Concentrate).

Looking at the target variable when using all the data at an hourly rate (Figure 3.7 on page 23), it is hard to notice any patterns, as there is simply too much noise. Therefore, it might be relevant to plot the respective rolling statistics, such as the moving average and the standard deviations above and below it – also known as Bollinger bands. By using the daily average, some of the noise is eliminated, but still, no patterns seem to emerge (Figure 3.8 on page 24).

Since the dataset contains about at least half a year's worth of observations, it is safe to try a rolling window of 168 observations, which is equivalent to 24 daily observations over a period of seven days (Figure 3.9 on page 24).

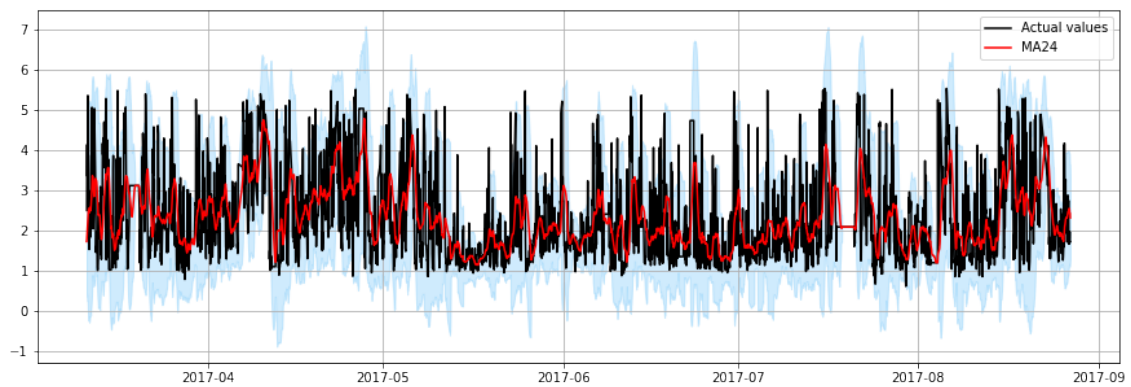


Figure 3.8: Plot with daily rolling statistics (moving average and Bollinger bands) for the target variable.

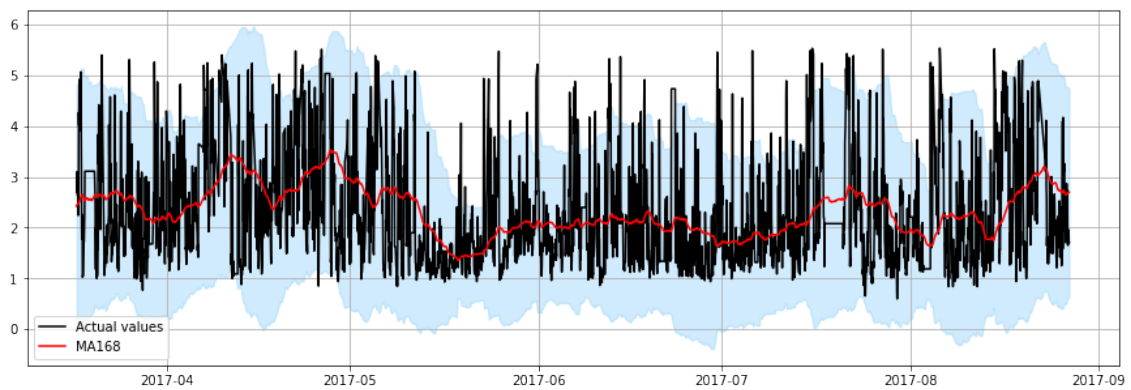


Figure 3.9: Plot with 7-day rolling statistics for the target variable.

Looking at the 7-day moving average, the pattern is still very irregular and noisy, but a slight downtrend starts to emerge. This movement is more easily noticed and confirmed by using a rolling window of 720 observations, i.e., 30 days (Figure 3.10 on page 24).

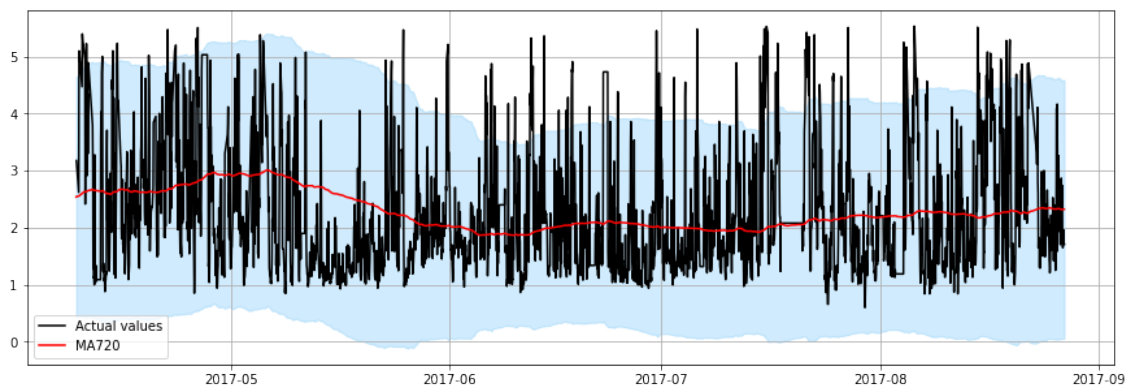


Figure 3.10: Plot with 30-day rolling statistics for the target variable.

3.5.3.4 Outlier Detection

An outlier is a data value that lies in the tail of the statistical distribution of a set of data values [13, 38]. After plotting the distribution of the target variable of both the hourly and daily values, it is easily observed that, in this case, there are not any significant outliers worth excluding.

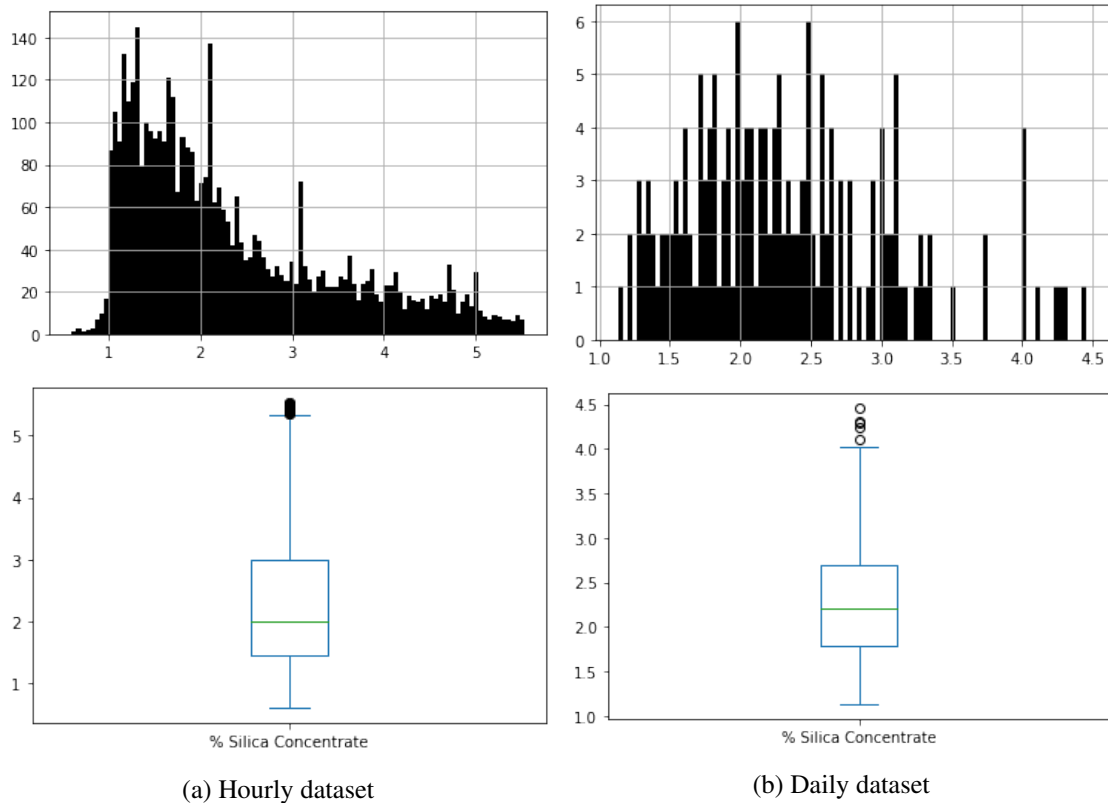


Figure 3.11: Histogram and box plot of the target variable's distribution of both the hourly and daily datasets.

3.5.3.5 Seasonality Analysis

The seasonal component is that part of the variations in a time series representing intra-year fluctuations that are more or less stable year after year with respect to timing, direction and magnitude [34, 38]. Understanding what kind of seasonality is affecting the data is necessary when building a forecasting model [38]. The `seasonal_decompose` function from the `statsmodel` Python module can be used to split the data into three components: trend, seasonality, and residuals. The best seasonal parameter is usually the one that minimizes residuals – the irregular component, which describes random, irregular influences. In this case, after some trial and error, the best values were obtained with a daily seasonality ($s = 24$), as remarked in Figure 3.12 on page 26.

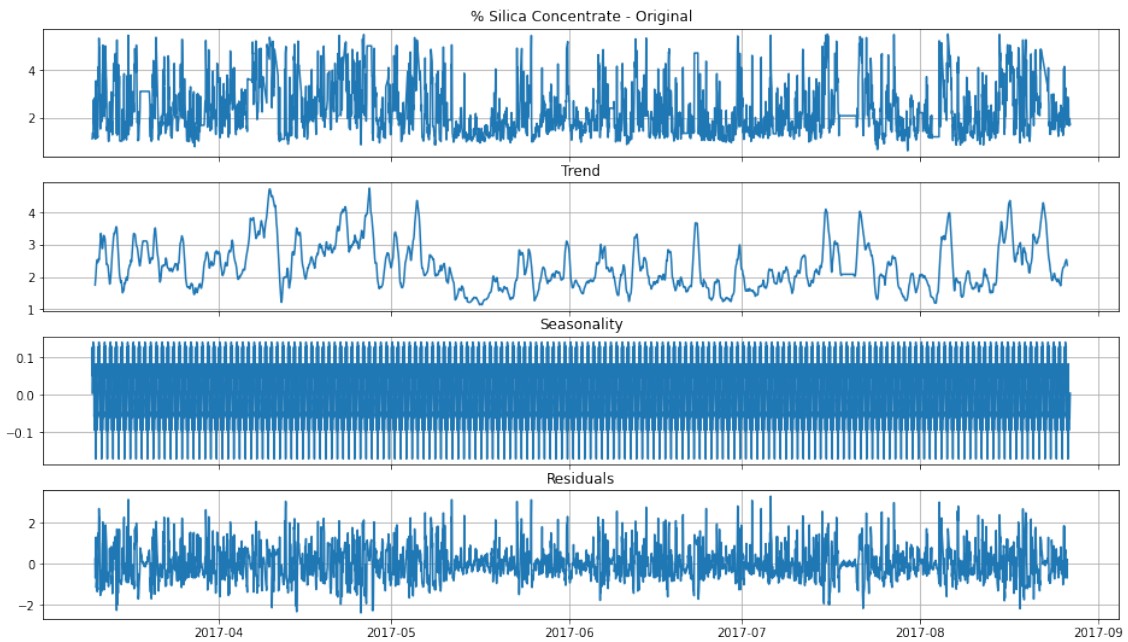


Figure 3.12: Time series decomposition: original data, trend, seasonality, and residuals.

3.6 Algorithms

3.6.1 Batch Learning

It was important to start by building a baseline batch learning model to be used for comparison with the online learning approach. Batch learning essentially consists in:

1. Loading the data;
2. Preprocessing the data;
3. Fitting the model to the data;
4. Compute the performance of the model on unseen data.

After loading the data and analyzing it, it was time to prepare the dataset that the models were going to use. Unwanted column features, as mentioned before, were removed, and the target variable (percentage of Silica Concentrate) was separated from the rest of the data. Moreover, it is critical to remember to scale the data. If there is a big difference in the range of the features, the model will make the underlying assumption that higher-ranging numbers have some sort of superiority [40]. Scaling makes it easier for the model to learn and understand the problem. Because the data is not normally distributed, it was best to use a MinMax Scaler – a scaler that scales the minimum and maximum values to 0 and 1, respectively.

Although this dataset comprises a highly autocorrelated time series, which means it would be important to keep the chronological integrity of the data, for the purpose of comparing the batch

and the online learning approach, the data was randomly split in training (70%) and testing (30%) datasets, so that the model could be trained and tested with different data. It was now possible to train the model by fitting it to the training data. The model consists of a simple linear regression on the features from the `scikit-learn` library. Finally, the testing set was used to make predictions and compute metrics to see how successful it was. In this case, the mean squared error (MSE) was used to tell how close the regression line was to the predicted points (Equation 3.3).

$$MSE = \frac{1}{n} \sum_{i=1}^n (f_i - y_i)^2 \quad (3.3)$$

The MSE metric takes the distances from the actual observed f_i values to the predicted y_i ones to calculate the errors and squares them in order to remove negative signs. The lower its final value, the better.

```
1 y_pred = reg.predict(X_test)
2 mse = mean_squared_error(Y_test, y_pred)
3
4 # 1.0723898311646687
```

3.6.2 Online Learning

Now that the baseline model was completed with reasonably good results ($MSE = 1.072$), it was time to build the online learning counterpart. The idea behind online learning, or, as the alternative name – incremental learning – suggests, is to fit the model to a stream of data. Unlike the batch learning approach, the data is not available in its entirety right away. Instead, observations are streamed one by one. `River`, as mentioned in Section 3.3, is designed to work with Python’s native dictionaries.

```
1 from river import stream
2
3 for x, y in stream.iter_pandas(df, df_target):
4     print(x, y)
5
6 # {'date': Timestamp('2017-03-10 02:00:20'), '% Iron Feed': 55.2, '% Silica Feed':
   16.98, 'Starch Flow': 3170.41, 'Amina Flow': 539.673, 'Ore Pulp Flow': 399.697,
   'Ore Pulp pH': 10.1589, 'Ore Pulp Density': 1.6691, 'Flotation Column 01 Air
   Flow': 249.291, 'Flotation Column 02 Air Flow': 248.269, 'Flotation Column 03
   Air Flow': 248.313, 'Flotation Column 04 Air Flow': 295.096, 'Flotation Column
   05 Air Flow': 306.4, 'Flotation Column 06 Air Flow': 250.488, 'Flotation Column
   07 Air Flow': 249.774, 'Flotation Column 01 Level': 462.601, 'Flotation Column
   02 Level': 488.724, 'Flotation Column 03 Level': 441.674, 'Flotation Column 04
   Level': 433.629, 'Flotation Column 05 Level': 448.477, 'Flotation Column 06
   Level': 480.866, 'Flotation Column 07 Level': 489.382, '% Iron Concentrate':
   67.06} 1.11
```

```
7 # ...
```

The way data is received in incremental learning mandates a different way to handle some aspects in relation to the batch setting. For instance, data scaling – an important aspect of obtaining good results – becomes tricky because to compute the mean and standard deviation, it is necessary first to pass over all of the data. This particularity completely defeats the purpose of incremental learning, which learns by looking at each observation only once and one at a time. Although this restriction seems to limit the usability of online learning, there are ways of overcoming these limitations and still reap benefits from this approach. `River` makes use of running statistics (or moving statistics) to estimate the mean and variance (and thus the standard deviation) to update itself when it is provided with a value. Even though these values will not be very accurate for the first few observations, as more data is streamed, these values become more accurate.

`River` makes it easy to build a simple pipeline to build the linear regression model. The pipeline can start with a `Select` method whose purpose is to choose the desired numeric features and discard the rest. Just like batch learning, a `MinMax Scaler` was added to scale the data that was fed to the linear regressor.

```
1 from river import evaluate
2 from river import metrics
3
4 evaluate.progressive_val_score(
5     model=model,
6     dataset=mining_dataset,
7     metric=metrics.MSE(),
8     print_every=50000
9 )
10
11 # [50,000] MSE: 0.008911
12 # [100,000] MSE: 0.007847
13 # ...
14 # [700,000] MSE: 0.005792
15 # MSE: 0.005772
```

The model was evaluated by using the MSE again, which was printed every 50,000 observations. The obtained result seemed too good to be true under real-world conditions, which could be explained by the way `progressive_val_score` method works. As explained in its documentation, as it is, this method is simply asking the model to predict the next observation, which might be only a few seconds ahead. Since the goal was to develop a model capable of dealing with real-world production conditions, to test it accordingly, it was necessary to also simulate such conditions. Only this way could the obtained results accurately attest the production behavior of the model. In this case, the intention was to forecast the percentage of Silica Concentrate 60 minutes ahead, which is the time it takes to measure it at the lab. Therefore, to simulate these conditions while building the model, the true percentage of Silica Concentrate could only be re-

vealed and used to update the model after the 60 minutes have passed. This can be done by setting the `moment` parameter of the `progressive_val_score` method – which determines which feature holds information about the timestamp of that observation – and the `delay` parameter – responsible for indicating how much time to wait until the true values are revealed to the model.

```

1 evaluate.progressive_val_score(
2     ...
3     moment='date',
4     delay=dt.timedelta(minutes=60)
5 )
6
7 # [50,000] MSE: 0.874913
8 # [100,000] MSE: 0.771805
9 # ...
10 # [700,000] MSE: 0.577828
11 # MSE: 0.574271

```

The score obtained ($MSE = 0.574$) after this change was not only much more realistic but also still an excellent result, given that the time series is highly autocorrelated.

3.6.2.1 Tweaking the Model

Feature Selection or attribute selection is a process where the best subset of features/attributes from the dataset are selected automatically. The concept of being the “best” is subjective and usually varies from problem to problem, but typically means higher accuracy. The task of selecting features can be seen as a state-space search, consisting of all possible combinations of attributes that can be chosen from the dataset. The end goal is to go through all of these and find the best, or at least a good enough, combination that improves performance in comparison to not selecting any subset of features. This can be done by using the `SelectKBest` module from `River` that selects a k number of features according to their scores. `SelectKBest`, by default, computes Pearson’s correlation between each feature and the target variable to make its selection. The main advantages of performing feature selection are the following:

- **Reduce Overfitting:** feature selection removes potentially redundant features, decreasing the likelihood of noise influencing the predictions;
- **Improve Accuracy:** less misleading data brings potential increases in accuracy;
- **Reduce Complexity:** fewer data inherently makes the algorithms faster.

It is important to look at the results (Table 3.8 on page 30) and take into account the absolute value of the scores when selecting the most relevant feature, as negative correlations are still relevant. Therefore, as expected, the percentage of Iron Concentrate had the highest score due to its high correlation with the target variable. As such, it was ignored once again. By looking at the

Table 3.8: Results of the feature selection using the `SelectKBest` module.

Feature	Score
% Iron Concentrate	-0.800423839059759
Flotation Column 01 Air Flow	-0.21965497525517683
Flotation Column 03 Air Flow	-0.21945331258778977
Flotation Column 05 Level	-0.1680547503714396
Flotation Column 02 Air Flow,	-0.16774411827477093
Amina Flow	0.15725827938410422
Flotation Column 04 Level	-0.14846789721914774
Ore Pulp pH	-0.14655450823036542
Flotation Column 07 Level	-0.14051868981110177
Flotation Column 06 Level	-0.10181641555724102
% Iron Feed	-0.07691126190443097
% Silica Feed	0.07230444713766719
Flotation Column 07 Air Flow	-0.07204373270089783
Starch Flow	-0.06746519909125351
Flotation Column 06 Air Flow	-0.04996067439607309
Ore Pulp Density	0.047053359515545845
Flotation Column 02 Level	0.03323913020321229
Flotation Column 01 Level	0.018896193414405054
Flotation Column 03 Level	0.015733121496269254
Flotation Column 05 Air Flow	-0.008701116263984192
Ore Pulp Flow	0.008567716336104817
Flotation Column 04 Air Flow	-0.0050445679089842245

remaining features and scores, it seemed reasonable to exclude those whose score was below 0.1, resulting in nine selected features out of twenty-two initial ones.

```

1 # [50,000] MSE: 0.864555
2 # [100,000] MSE: 0.766691
3 # ...
4 # [700,000] MSE: 0.572935
5 # MSE: 0.569464

```

Now that the score has improved slightly after performing feature selection ($MSE = 0.569$), it could be further tweaked by finding the optimal model architecture. This was achieved by performing hyperparameter tuning, i.e., selecting the optimal parameters that define the model architecture (also known as hyperparameters). This is usually done automatically and exhaustively by generating all possible combinations from a given grid of parameter values – grid searching. The grid, in this case, consisted of three different optimizers: Adam, plain Stochastic Gradient Descent (SGD), and RMSProp – a mini-batch version of the resilient backpropagation (RProp) heuristic [22]. For all of them, various values of the learning rate `lr` were tested and, in the case of Adam, also different initial decay rates `beta_1`.

```

1 models = utils.expand_param_grid(model, {
2     'LinearRegression': {
3         'optimizer': [
4             (optim.SGD, {'lr': [.2, .15, .1, .01, .005, .001]}),
5             (optim.RMSProp, {'lr': [.2, .15, .1, .01, .001]}),
6             (optim.Adam, {'beta_1': [.9, .5, .2, .1, .01], 'lr': [.2, .15, .1, .01,
7                             .001]}),
8         ]
9     }
10 })

```

An halving algorithm, for every iteration, only retains the consistent experts – the predictor who gives the correct label for all instances – dropping predictors that make mistakes. Another advantage of this model selection method is that it does not require training each model on all data. However, it was not worth it in this case because River’s `SuccessiveHalvingRegressor` chose the same hyperparameters. Therefore, an alternative way of improving the model’s score was attempted by performing ensemble learning – an approach that seeks better predictive performance by combining multiple models.

```

1 ensemble.EWARRegressor([
2     linear_model.LinearRegression(optim.SGD()),
3     linear_model.LinearRegression(optim.RMSProp()),
4     linear_model.LinearRegression(optim.Adam())
5 ])
6
7 # [50,000] MSE: 0.864102
8 # [100,000] MSE: 0.766324
9 # ...
10 # [700,000] MSE: 0.572687
11 # MSE: 0.566415

```

River’s `EWARRegressor` from the expert module can be used to combine three different linear regression models, each trained with different optimizers. The goal was to run them in parallel and, according to their individual performance, assign them weights that are used to make more accurate predictions. Using ensemble learning worked as expected because it ended up slightly improving the model ($MSE = 0.566$).

3.7 Deployment

While Jupyter notebooks are great for prototyping the data workflow, testing different approaches, and drawing interesting conclusions, they cannot emulate the production environment and real-world conditions that this framework intended to test. To achieve this, it was crucial to not only build the models but also deploy them so to obtain a fully functional service.

To achieve this, a straightforward solution was to use an API to serve the model. An Application Programming Interface (API) is a set of definitions and protocols for building and integrating application software [20]. In other words, it is a web service that acts as an intermediary when two applications need to communicate via standard HTTP protocols. However, an API, by itself, does not have a frontend, i.e., a graphical interface, for ease of use.

To tackle this drawback, the perfect approach was to deploy the machine learning model as a web application by wrapping the API via the use of web frameworks such as Django and Flask. As the prime goal was to build a simple and lightweight backend service, Flask was used instead of Django, as it enables a rapid and easy development of a fully functioning REST API. To keep the frontend development equally simple and lightweight, Streamlit¹¹, a low-code open-source Python library, was used. Streamlit itself uses React¹² as a frontend framework to render the data on the screen, but it makes the app creation process as simple as writing a Python script that seamlessly integrates with other popular python libraries such as NumPy, Pandas, Scikit-learn, and RiverML.

3.7.1 User Interface

The framework's interface allows the user to try different combinations of settings for the two available models: online (Figure 3.12 (a) on page 26) or batch learning (Figure 3.12 (b) on page 26).

For both models, any of the 22 columns can be simultaneously dropped, and three different metrics – mean absolute error (MAE), root mean square error (RMSE), and symmetric mean absolute percentage error (SMAPE) – can be tracked. The anomaly detection setting lets the user choose the feature to be analyzed and what detector to use. Specifically for the case of the online learning model, a given delay – in seconds – can be chosen, dictating how much time it should take to reveal to the model the true target value of the current observation. This obviously does not apply to the batch learning model. Instead, the size of the training set can be indicated. The ability to tinker with the models makes carrying out different experiments straightforward and intuitive.

3.7.2 Gateway Simulator

To simulate a production environment, a simulated gateway was developed in Python whose sole purpose is to stream data to the server (backend). This simulates what a physical device, for instance, an industrial machine containing different sensors, would do in real-life. The frequency of the stream can be changed, but by default, it sends a POST request to the `/stream` endpoint every one second.

3.7.3 API

Often resources have one or more methods that can be performed on them over HTTP, like GET, POST, PUT, and DELETE. This API uses only two of them: GET to view information without

¹¹<https://streamlit.io/>

¹²<https://reactjs.org/>

(a) Online learning settings

(b) Batch learning settings

Figure 3.13: Framework's user interface settings.

changing it and `POST` for changing information on the server. Both the batch and online learning models have individual endpoints to be initialized and have their current (updated) metrics read.

3.7.3.1 `/stream`

The Flask server, as previously mentioned, has the `/stream` endpoint, which is responsible for receiving the raw sensor data from the gateway simulator. It consists of a `GET` method without any parameters that returns a `success` message when the data is successfully received. In case any of the models have been initialized, receiving this new data triggers the process of updating them. The batch model is never updated unless the model has been, besides initialized, also trained, which happens when the total number of received observations becomes greater than the set minimum amount of data (specified by the user when the model is initialized). Otherwise, both models' metrics are updated after using the newly received observation to make predictions on the target variable. Unlike the batch learning model, which is only trained with the user-set number of training observations, the incremental learning model learns from each incoming observation with a default learn delay of zero.

3.7.3.2 `/batch/train`

The batch model is initialized and trained every time there is a GET request to the `/batch/train` route. It has two required parameters: `columns_to_drop` – the features from the data that are to be dropped – and `train_size` – the number of observations to be used for training. If that condition is fulfilled (more data has been streamed than the defined amount), that exact quantity of data is scaled and fitted to a linear regression model. Both the scaler and regressor are saved to be used for predictions. It also accepts two optional parameters: `anomaly_detector` – the detector to be used to perform anomaly detection – and `anomaly_feature` – indicating the feature to be used for the anomaly detection property. Finally, if successful, it returns the time it took for the model to be trained.

3.7.3.3 `/batch/metrics`

This endpoint, just as the name suggests, following a GET request, returns the current (updated) values of the three available metrics – MAE, RMSE, and SMAPE – as a JSON object. In case anomaly detection is ongoing, it also returns that result for the most recent observation – a value of zero in case it was classified as a regular data point or one in case it was labeled as anomalous. The true target value for that same observation is also returned so that its evolution can be displayed, and the correlation between them and the anomalies can be computed.

3.7.3.4 `/online/init`

Unlike the batch learning counterpart, the streaming model does not require training. As such, the model is immediately built and saved for later prediction whenever there is a GET request to the `/online/init` route. It has two required parameters: `columns_to_drop` – similar to the batch learning one – and `learn_delay` – which dictates how much time it is required before the true target value of that observation is revealed to the model, i.e., the lag with which the model is allowed to use that sample to learn. It also accepts `anomaly_detector` and `anomaly_feature` as optional parameters.

3.7.3.5 `/online/metrics`

In everything similar to the `/batch/metrics` endpoint.

Chapter 4

Experiments and Results

This chapter demonstrates the results obtained with the different experiments, both individually and working together as a framework.

Section 4.1 presents the experiments done for both the Anomaly Detection module (Section 4.1.1) and Time Series Forecasting (Section 4.1.2) individually. At last, Section 4.2 explains how the different modules were merged under the same framework and the results it yielded.

4.1 Experiments

4.1.1 Anomaly Detection

As mentioned in the Implementation (Chapter 3), NAB's dataset was initially used to benchmark and compare the different online anomaly detectors and assess how well they detect anomalies. Taking into account the methods available on `River` and the data being analyzed, two different detectors were tested:

- **Half-Space Trees:** online variant of isolation forests. They work well when anomalies are spread out, in contrast to anomalies that are packed together in windows;
- **One-Class SVM:** unsupervised kernel-based anomaly detection method.

These detectors compute a score for each observation, which represents the likelihood of being an anomaly. High scores indicate anomalies, whereas low scores indicate expected, regular observations. Therefore, it is fundamental to set a threshold above which to classify a given score as probably anomalous:

- **Constant Thresholder:** each anomaly score is thresholded into a 0 or a 1;
- **Quantile Thresholder:** each anomaly score is thresholded according to the value of the chosen quantile.

After some adjustments, the results displayed in Table 4.1 on page 36 were obtained.

Table 4.1: Anomaly detection results. (1) $n_trees=10$, $height=5$, $window_size=250$, $threshold=0.9$; (2) $n_trees=7$, $height=9$, $window_size=250$, $threshold=0.85$; (3) $nu=0.5$, $threshold=0.7$; (4) $nu=0.5$, $threshold=0.8$.

F1	Half-Space Trees		One-Class SVM	
	Constant Threshold	Quantile Threshold	Constant Threshold	Quantile Threshold
	(1) 50.43%	(2) 50.35%	(3) 39.71%	(4) 42.66%

With the Half-Space Tree detector obtaining promising results on the control data, it was time to test it on the real-world data. Looking at the Flotation Column 01 Air Flow feature, it becomes an excellent candidate to test the anomaly detector as it is easy to distinguish visually the expected values from the noise (Figure 4.1 on page 36).

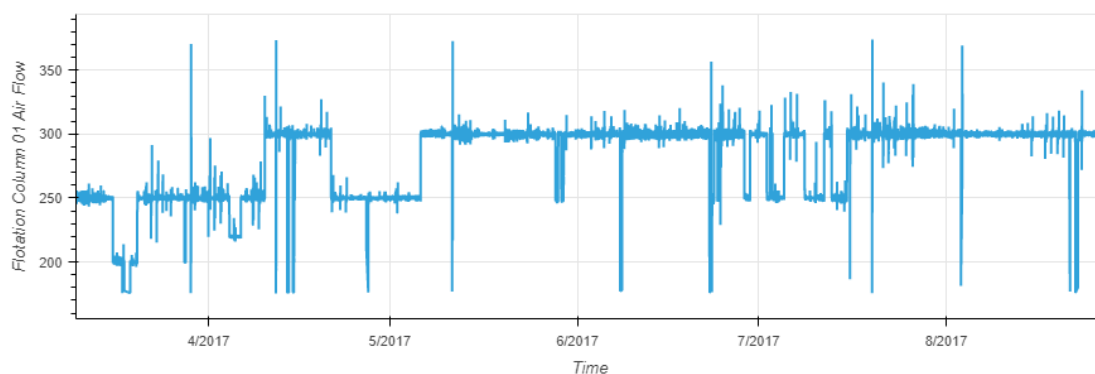


Figure 4.1: Plot of the Flotation Column 01 Air Flow feature.

To make it more evident, excess noise can be removed by computing the hourly average for this feature (Figure 4.2 on page 36). This smooths out the values by removing most of the anomalies.

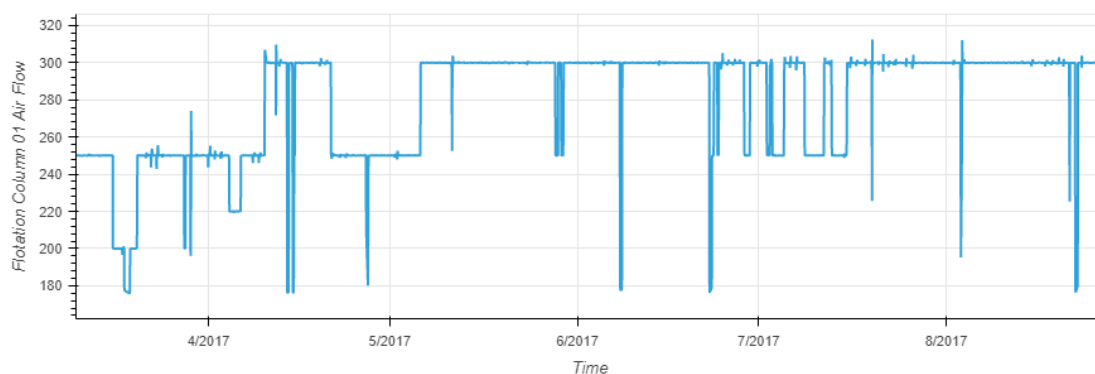


Figure 4.2: Plot of the hourly average for the Flotation Column 01 Air Flow feature.

As seen in Figure 4.2 on page 36, the detector worked fairly well and detected most of

the anomalies. In addition, the same anomaly detection task can be performed using the well-established offline approaches available from the `scikit-learn` library. As previously mentioned, such machine learning techniques lack the challenges that streaming data introduce even when doing anomaly detection, which requires special algorithms like Streaming Half-Space Trees [42] in order to maintain high detection accuracy. With this being said, the results of the `scikit-learn` implementation of the One-Class SVM can be seen on Figure 4.4 on page 37.

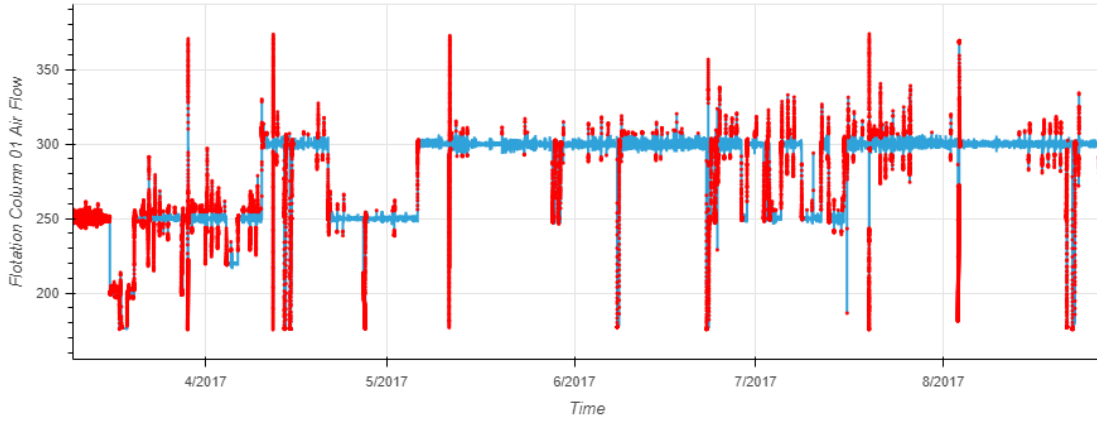


Figure 4.3: Detected anomalies (in red) for the Flotation Column 01 Air Flow feature using the Half-Space Tree Detector with a constant threshold for the online learning approach.

Although the results of the offline detector seem to be slightly better at detecting finer drifting, the time to compute them was considerably longer.

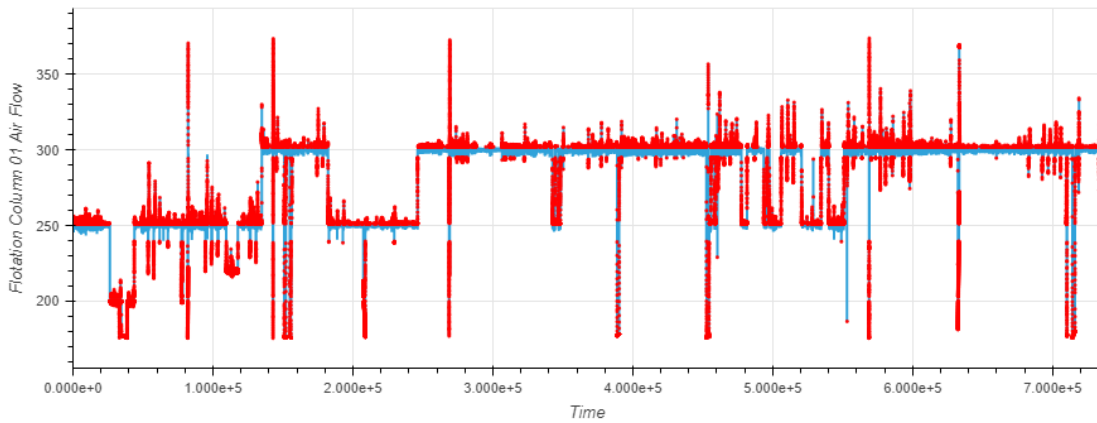


Figure 4.4: Detected anomalies (in red) for the Flotation Column 01 Air Flow feature using the One-Class SVM Detector from the `scikit-learn` library for the batch learning approach.

At last, the labeled anomalies derived from the Flotation Column 01 Air Flow feature were used to compute their correlation with the target variable (percentage of Silica Concentrate) to understand if they explain some of the variations on the impurity level (Figure 4.5 on page 38).

Interestingly, the anomalies detected by the online model seem to explain the target variable better, perhaps because it was less strict at distinguishing anomalies from normal-occurring observations.

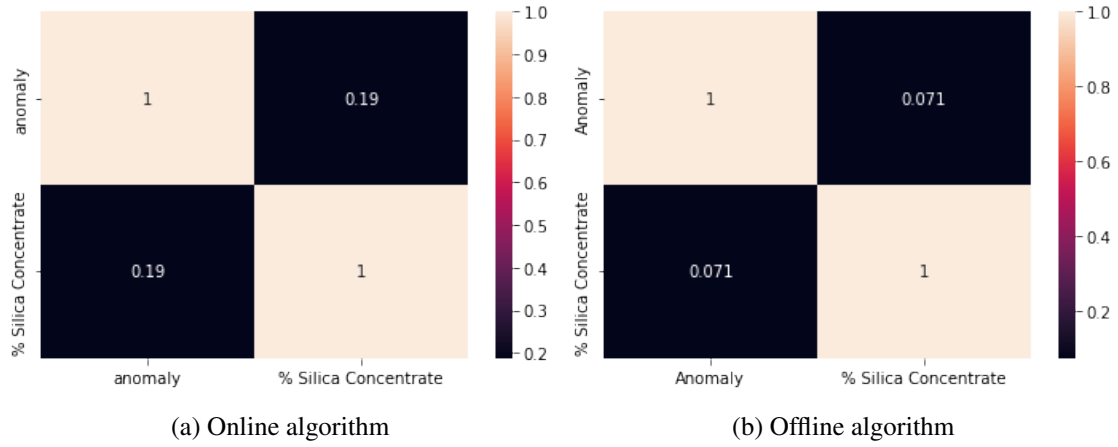


Figure 4.5: Correlation matrix between the detected anomalies for the Flotation Column 01 Air Flow feature and the target variable (% Silica Concentrate).

4.1.2 Time Series Forecasting

The ARIMA model – one of the most common methods – was used to perform time series forecasting after being fitted to some data points of that series. There is a different parameter for each of the three different ARIMA terms (AR, I, and MA). In the case of the AR, the p parameter indicates the number of lagged values that are going to be used for the forecasting. For the I, the d parameter sets the differencing order necessary to make the series stationary. Finally, for the MA, the q parameter informs about the amount of forecast error terms to use in the prediction. Since the time series is stationary (as per the time series analysis), the d value will be set to zero. To determine the p and q values, both the ACF and Partial Autocorrelation Function (PACF) were used. Detailed plots are shown on Figure 4.6 on page 38.

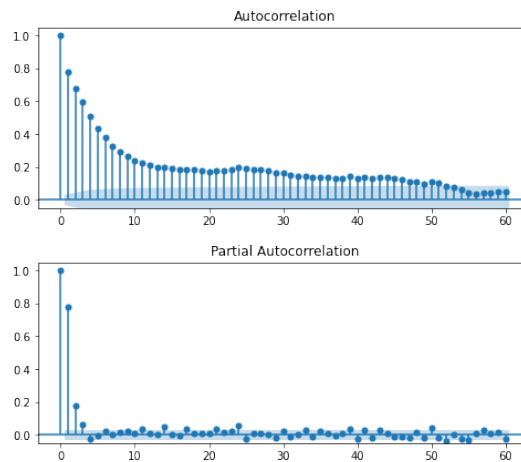


Figure 4.6: Detailed plots of the ACF and PACF for the first 60 lags.

The ACF plots the correlation between an observation and its lagged value. Values outside the 95% confidence interval defined by light blue bands indicate a strong correlation. On the other hand, the PACF captures the correlation between the time series with a lagged version of itself, ignoring insignificant time lags that do not have an effect on the future or present time period. For p , spikes outside the blue bands on the ACF plot are potential good values. The same goes for determining the order of q on the PACF. As such, AR(25) and MA(2) can be chosen as starting points. Having determined possible values for the (p, d, q) parameters, it was possible to build the ARIMA model.

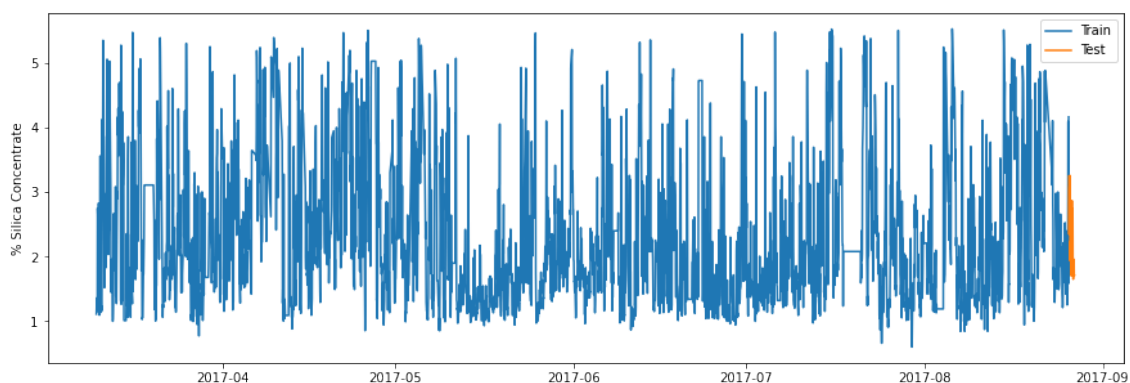


Figure 4.7: Plot of the target variable's (% Silica Concentrate) training and test data for the ARIMA's forecasting.

Since ARIMA is usually superior for short-term forecasting, it was used to forecast a day of the time series. To test its accuracy, the last 24 hours of the series were removed from its training data and used (as shown in Figure 4.7 on page 39 and in detail on Figure 4.8 on page 39).

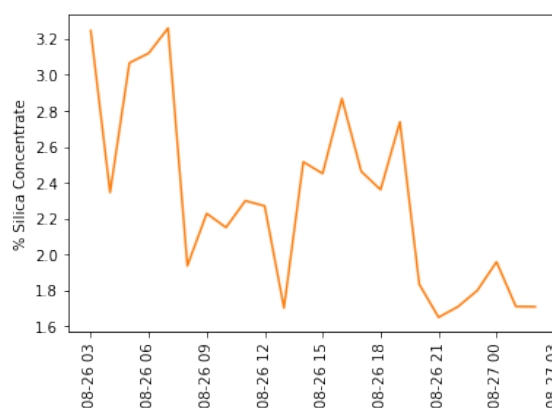


Figure 4.8: Plot of the 24-hour interval to be forecasted by the ARIMA.

By implementing an ARIMA model and fitting it to the $(p = 25, d = 0, q = 2)$ parameters, the forecast on Figure 4.9 on page 40 was obtained. It is evident the model did not catch up with the highs and lows of the original series, but it captured the essence of the data without overfitting it.

Some further experimentation with other p and q values could be a way of improving the model if necessary.

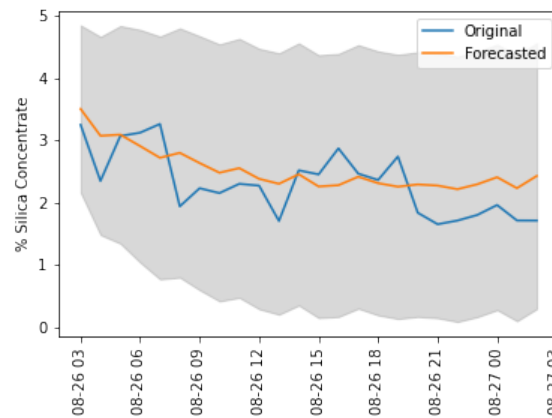


Figure 4.9: Comparison of the original data (blue) with the ARIMA's forecast results (orange).

Another way of performing forecasting would be to use auto-ARIMA, which makes use of information criterions like the Akaike Information Criterion (AIC) – a metric of the quality of a given statistical model – to tune its hyperparameters. The only problem of this approach is the lack of explainability of the parameters, as it is good practice to be able to explain their values and impact on the predictions. For the same test, auto-ARIMA automatically adjusted to ($p = 2$, $d = 0$, $q = 1$) and made the predictions on Figure 4.10 on page 40.

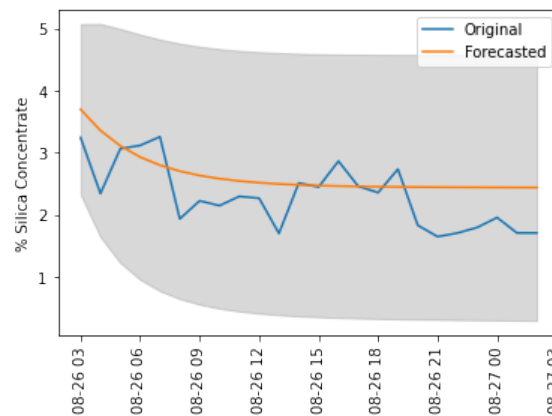


Figure 4.10: Comparison of the original data (blue) with the auto-ARIMA's forecast results (orange).

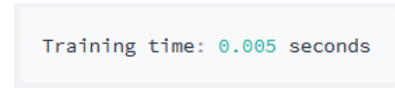
Having lower values for both p and q causes the model to be faster, as the granularity of the predictions is also lower. Nonetheless, the model still captured the essence of the data accurately.

4.2 Framework

After completing the experiments with the individual modules, all but the forecasting one, were merged together under a framework as described in the Architecture (Section 3.2). It provides the ability to change the models' settings and configurations through a dashboard to see the impacts in real-time. In the case of the batch learning model, the framework allows the user to set the number of observations to use for training. The number of available samples received is updated and displayed (Figure 4.11 (a) on page 41). When that threshold is reached, the model is trained with those observations, and the time it took to do so is shown (Figure 4.11 (b) on page 41).



(a) Train samples setting



(b) Training time message

Figure 4.11: Batch learning training.

In this example, a training size of 94 samples was used. The target value (% Silica Concentrate) changes every 180 observations, which corresponded to a lab measurement every hour in the initial dataset. This circumstance is analogous to natural-occurring concept drift and highlights the main weakness of the offline learning approach. As seen on Figure 4.12 on page 41, after the model learns from the first 94 samples, the error for the following 86 (180 - 94) is zero. However, when the target value changes, the error increases instantly as the model does not know how to explain such change.

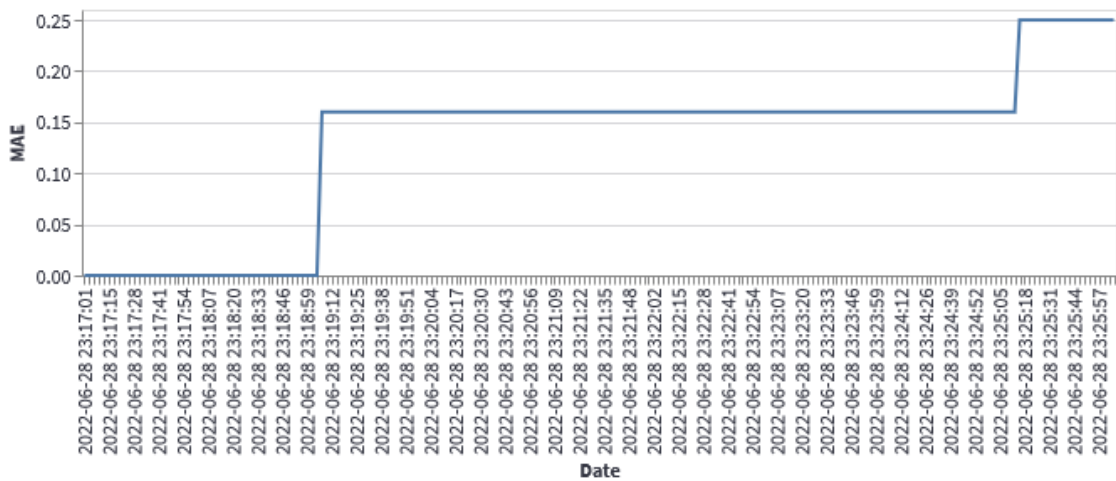


Figure 4.12: MAE for the first few observations after training the batch regressor with 94 samples.

Obviously, 94 samples for training is a very small number that was only used to demonstrate the underlining problem. For more accurate results, more training samples are required. As an example, Figure 4.13 on page 42 is the results after having the model retrain with 555 samples.

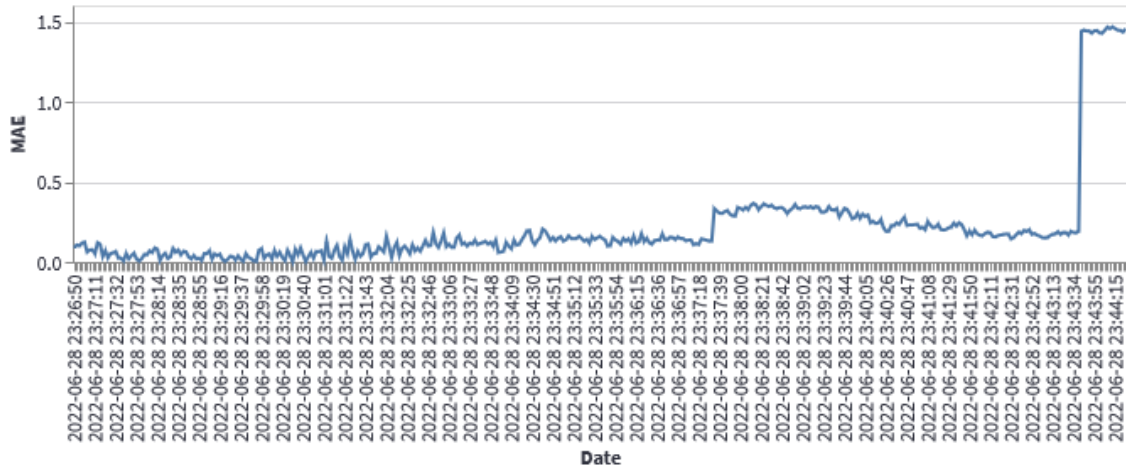


Figure 4.13: MAE after training the batch regressor with 555 samples.

Initially, the results seem very promising, as the error is relatively low and seems to adapt well to drifting. Nevertheless, it is only a matter of time until an unexpected reading is received and error hikes. This demonstrates how batch learning is hugely dependent on regular retraining to fight back concept drift, which the framework allows at the user's command when the number of samples needed for training is changed. Comparing to the online learning approach, behavior is entirely different. Online learning "trains", i.e., learns, with each observation. This implies that initially, the error is more prominent and gets lower throughout time as more data is streamed (Figure 4.14 on page 42).

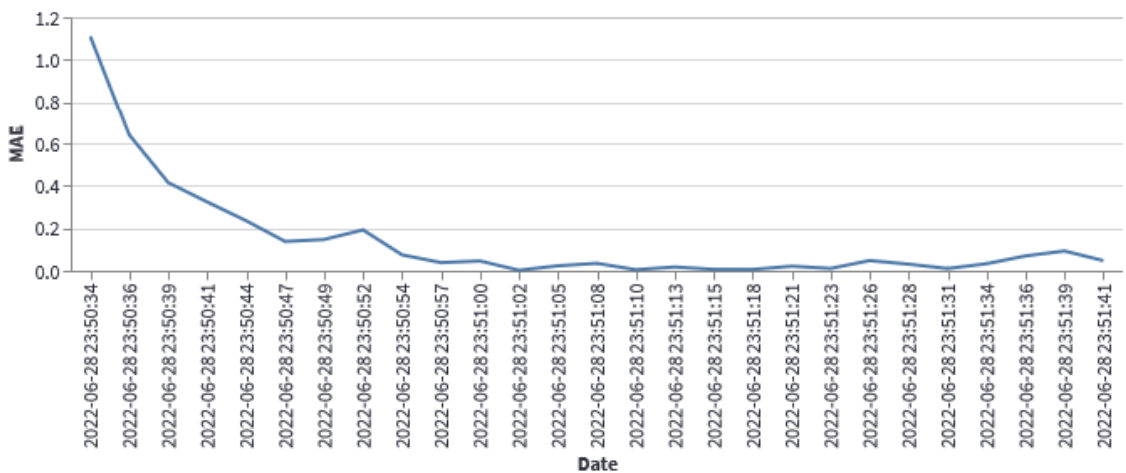


Figure 4.14: MAE of the first few observations for the online regressor.

Unlike its batch learning counterpart, it is never necessary to retrain the streaming model. Even when drifting occurs, the model quickly learns and adapts to that variability (Figure 4.15 on page 43).

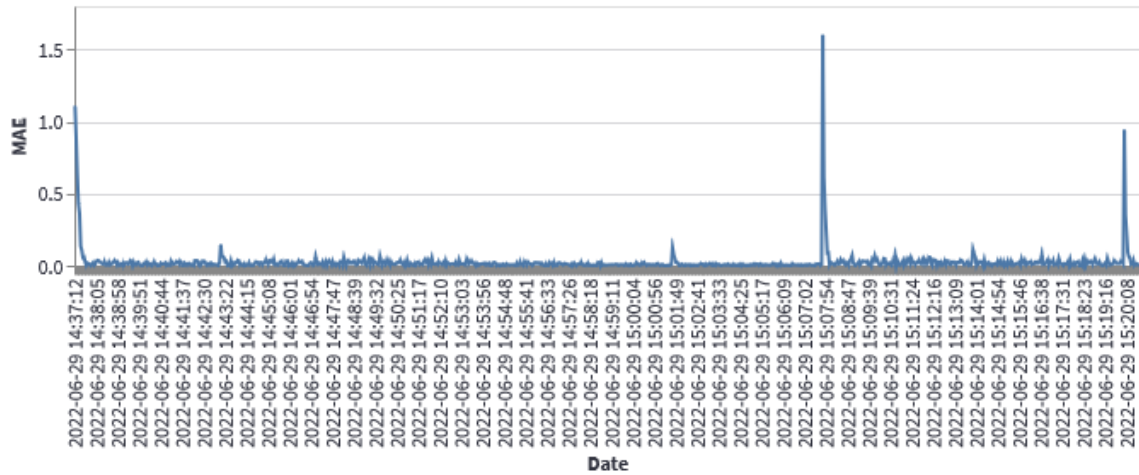


Figure 4.15: MAE hikes after some more observations for the online regressor.

Another feature of the framework comes from the anomaly detection module and adds the ability to perform anomaly detection on any chosen column variable with the chosen anomaly detector and threshold. Detected anomalies are labeled as such, which allows the model to learn from that and improve predictions. When a given feature is chosen – in this example, the Flotation Column 01 Air Flow variable – its streamed values are displayed so that they can be visually interpreted (Figure 4.16 on page 43).

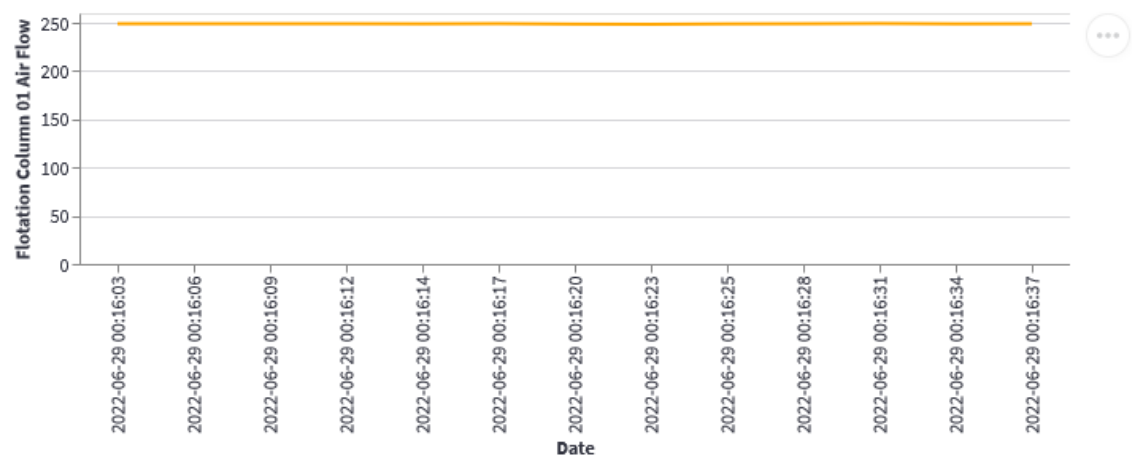


Figure 4.16: Evolution of the Flotation Column 01 Air Flow feature throughout time (chosen for anomaly detection).

Initially, there is little to no variation, which evidently means there is a very low probability of

an anomaly being detected. Therefore, there cannot be any correlation between these anomalies and the target variable. Figure 4.17 on page 44 expectedly displays that exact lack of anomalies and correlation.

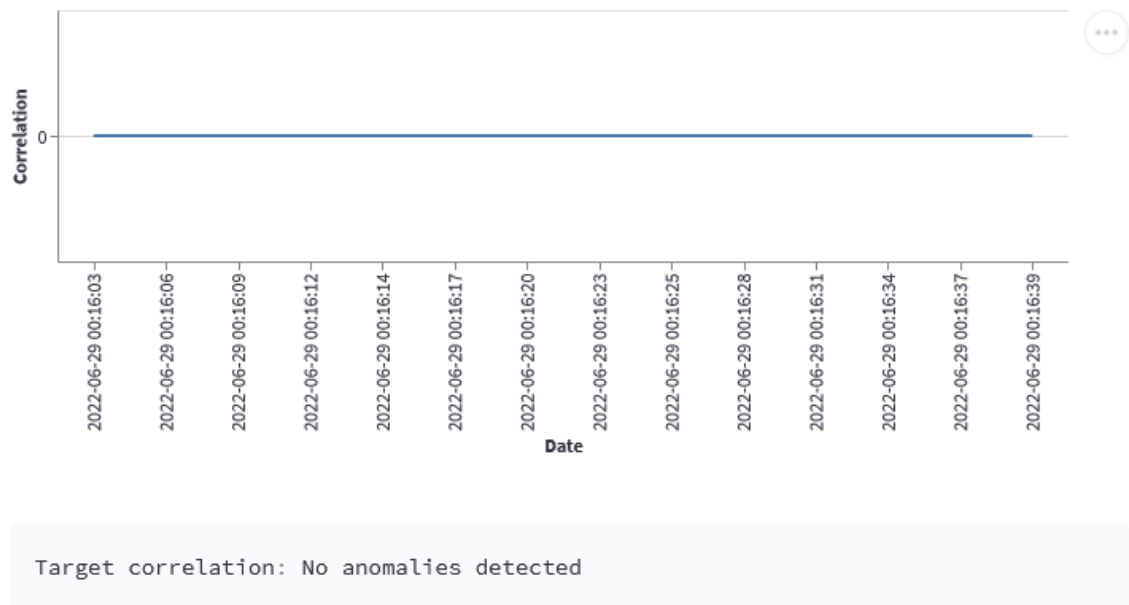


Figure 4.17: Plot and feedback message of the target correlation while no anomalies have been detected.

As time passes and Flotation Column 01 Air Flow's values vary, some anomalies are detected and correlations with the target are updated (Figure 4.18 on page 44).

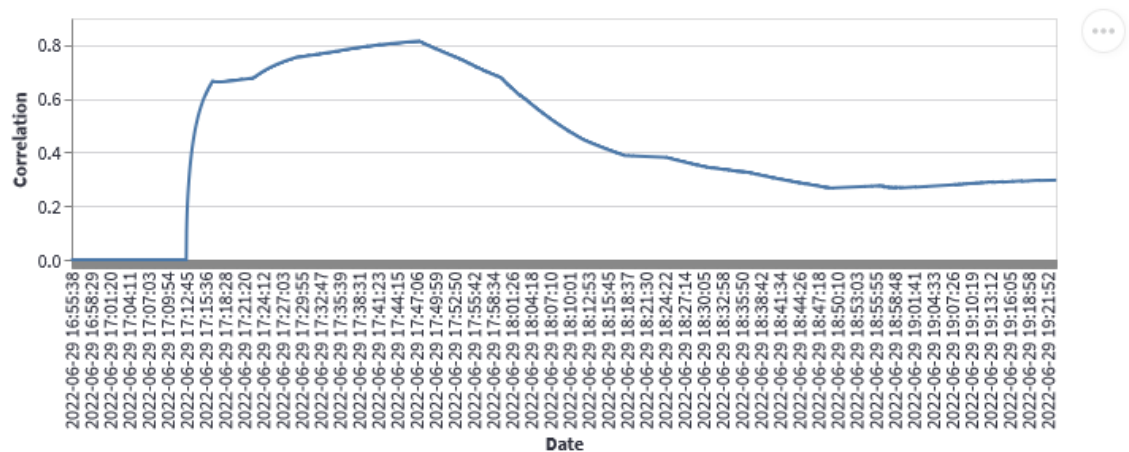


Figure 4.18: Plot of the correlation between the detected anomalies and the target variable.

Chapter 5

Conclusions and Future Work

This chapter wraps up the work completed for this dissertation.

Here the main takeaways and contribution from this dissertation's work will be presented (section 5.1), as well as future work that could be developed on top of this project to further improve it (section 5.2).

5.1 Conclusions

Industry 4.0 is responsible for generating increasingly larger amounts of data that become crucial during manufacturing processes. Given that traditional machine learning models were not initially conceived to handle the velocity, volume and variety that Big Data brought, they have become a potential risk for the industry. This is due to the fact that such models assume data is static. By doing so, they ignore unavoidable drifting that eventually renders them obsolete unless retrained with new data.

Literature indicates a more novel and effective way to overcome this flaw is by using online learning algorithms that continuously learn and adapt to every new data sample.

This dissertation focused on exploring exactly this approach by building a streaming process framework that is capable of managing the traditional ML algorithms' life cycle, detecting data drift of some key feature variables (for instance, from sensor anomalies or manufacturing quality degradation), and deliver adaptive predictions of some indicators' behavior (for instance, to forecast how a given product quality will evolve).

The framework comprises a traditional batch learning and a streaming online model module. These employ, respectively, simple `scikit-learn` and `RiverML` Linear Regression models to make predictions on a product's final quality given the input of a manufacturing machine. The batch learning approach proved, as expected, to be prone to performance degradation when data drift occurs. Nonetheless, since its performance is monitored, it can be retrained once a given threshold is reached. On the other hand, the online learning approach lived up to the expectations

and demonstrated high adaptation capabilities to drift. Additionally, after promising control tests, the anomaly detection module was added with different detectors (such as the One-Class SVM). Its purpose is to track concept drift of any given data feature. These anomalies are taken into account to compute their correlation with the target variable in order to explain the variations of the results, i.e., to differentiate, for example, if the drift might be caused by wrong sensor readings or actual manufacturing quality degradation. A forecasting model was also developed to forecast some of the metrics' behavior throughout time, like the evolution of the final product quality.

The key innovative aspect of this work is the ability to tinker with the models' settings and configurations through the dashboard to see the impacts in real-time. This makes carrying out different experiments straightforward and intuitive. Besides the practical exploratory analysis of this issue, this dissertation aimed to develop a production-ready framework deployed to a production-like environment as a web application built on Streamlit with its own Flask API. The data arriving as input was not from an actual machine, but a gateway simulator was used to continuously stream data as a real machine sensor would. This platform was developed and will continue to be improved and updated by DIGI2 lab¹. Ultimately, it is intended to be used in research projects such as RECLAIM² and DEZMPP³. In addition, an article based on the work developed for this dissertation will be submitted for review to the 12th International Conference on the Internet of Things⁴.

5.2 Future Work

Although the results of this dissertation provide a robust streaming framework for adaptive data analytics, there are still a few aspects that could be improved. For starters, the tests were conducted with a prepared dataset containing data from an iron ore mine that was being streamed from a gateway simulator. Despite the fact that this gateway was reproducing a production environment, testing it with raw real-time data coming from an actual machine sensor would be ideal to validate the results further.

Having a true production environment in mind with resource-intensive requirements, it would probably be beneficial to migrate the web app to a fully-scalable web framework like Django. For the purpose of this dissertation, Django provided excessive overhead, with Streamlit in combination with Flask proving to be a better option for simplicity and ease of development.

In terms of the capabilities of the framework, it would also be interesting to provide more precise control over the settings and parameters of each module. For instance, the anomaly detection feature allows the user to select different detectors and thresholds. However, it could be upgraded to allow for hyperparameter changes of the detectors themselves, for example, the `n_trees` parameter of the Half-Space Tree anomaly detector. The same goes for many other

¹<http://digi2.fe.up.pt/>

²<https://www.reclaim-project.eu/>

³<https://dezmpp.zeugma.pt/>

⁴<https://iot-conference.org/iot2022>

hyperparameters of each of the other modules. This added configurability would allow for more extensive experimentation with the models.

Finally, the forecasting module could be extended to provide real-time insights on the framework of how the model makes decisions, i.e., forecast interpretability.

References

- [1] S. Alfayoumi. Anomaly Detection. <https://ksp-windmill-itn.eu/research/anomaly-detection/>, 2021. Accessed: 2022-06-01.
- [2] M. Baena-García, J. Campo-Ávila, R. Fidalgo-Merino, A. Bifet, R. Gavaldà, and R. Morales-Bueno. Early Drift Detection Method. In *Proceedings of the ECML PKDD International Workshop on Knowledge Discovery from Data Streams (IWKDDs'06)*, pages 77–86, January 2006.
- [3] A. Bifet and R. Gavaldà. Learning from Time-Changing Data with Adaptive Windowing. In *Proceedings of the 7th SIAM International Conference on Data Mining*, volume 7, pages 443–448, April 2007.
- [4] L. Bontemps, V. L. Cao, J. McDermott, and N. Le-Khac. Collective Anomaly Detection based on Long Short Term Memory Recurrent Neural Network. *CoRR*, 2017.
- [5] G. E. P. Box and G. M. Jenkins. *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, CA, 1970.
- [6] L. Breiman. Random Forests. *Machine Learning*, 45:5–32, October 2001.
- [7] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Comput. Surv.*, 41(3):1–58, July 2009.
- [8] A. A. Cook, G. Misirli, and Z. Fan. Anomaly Detection for IoT Time-Series Data: A Survey. *IEEE Internet of Things Journal*, 7(7):6481–6494, 2020.
- [9] D. Dickey and W. Fuller. Distribution of the Estimators for Autoregressive Time Series With a Unit Root. *JASA. Journal of the American Statistical Association*, 74, June 1979.
- [10] J. Ding, J. Fang, P. Chao, J. Xu, P. Zhao, and L. Zhao. A Distributed Framework for Online Stream Data Clustering. In *Algorithms and Architectures for Parallel Processing. Proceedings of the 20th International Conference (ICA3PP 2020)*, volume I, pages 190–204, Cham, Switzerland, 2020.
- [11] P. Domingos and G. Hulten. Mining High-Speed Data Streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, page 71–80, New York, NY, USA, 2000. Association for Computing Machinery.
- [12] D. Dua and C. Graff. UCI Machine Learning Repository, 2017.
- [13] Economic Commission for Europe of the United Nations (UNECE). Glossary of Terms on Statistical Data Editing. In *Conference of European Statisticians Methodological material*, Geneva, 2000.

- [14] J. Gama, P. Medas, G. Castillo, and P. Rodrigues. Learning with Drift Detection. In *Intelligent Data Analysis*, volume 8, pages 286–295, September 2004.
- [15] H. M. Gomes, A. Bifet, J. Read, J. Paul Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106:1469–1495, 2017.
- [16] H. M. Gomes, J. Read, and A. Bifet. Streaming Random Patches for Evolving Data Stream Classification. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 240–249, November 2019.
- [17] H. M. Gomes, J. Read, A. Bifet, J. P. Barddal, and J. Gama. Machine learning for streaming data: state of the art, challenges, and opportunities. *ACM SIGKDD Explorations Newsletter*, 21(2):6–22, December 2019.
- [18] F. E. Grubbs. Procedures for Detecting Outlying Observations in Samples. *Technometrics*, 11(1):1–21, 1969.
- [19] D. Hand and R. Till. A Simple Generalisation of the Area Under the ROC Curve for Multiple Class Classification Problems. *Hand, The*, 45:171–186, November 2001.
- [20] Red Hat. What is a REST API? <https://www.redhat.com/en/topics/api/what-is-a-rest-api>, 2020. Accessed: 2022-05-29.
- [21] D. M. Hawkins. *Identification of Outliers*. Springer, Dordrecht, U.K., 1980.
- [22] G. Hinton. Neural Networks for Machine Learning. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Accessed: 2022-06-20.
- [23] A. Dotis-Georgiou (InfluxDB). Autocorrelation in Time Series Data. <https://www.influxdata.com/blog/autocorrelation-in-time-series-data/>, 2019. Accessed: 2022-06-01.
- [24] Information and Computer Science University of California. KDDC99. <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>, 1999. Accessed: 2022-02-22.
- [25] P. T. T. Khine and H. P. P. Win. Ensemble Framework for Big Data Stream Mining. In *2020 IEEE Conference on Computer Applications (ICCA)*, page 5, Piscataway, NJ, USA, 2020.
- [26] J. Kurtz and R. Shockley. Analytics: The real-world use of big data in manufacturing. Technical Report GBE03570-USEN-00, IBM Institute for Business Value, Somers, NY, U.S.A., July 2013.
- [27] P. Li, X. Wu, and H. Xuegang. Learning from Concept Drifting Data Streams with Unlabeled Data. In *Neurocomputing*, volume 92, January 2010.
- [28] B. Lim and S. Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A (Mathematical, Physical and Engineering Sciences)*, 379(2194):14, April 2021.
- [29] G. Louppe and P. Geurts. Ensembles on Random Patches. In *Proceedings of the 2012 European conference on Machine Learning and Knowledge Discovery in Databases*, pages 346–361, September 2012.

- [30] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and Guangquan Z. Learning under Concept Drift: A Review. *arXiv*, page 18, April 2020.
- [31] T. M. Mitchell. *Machine Learning*, 1997.
- [32] P. A. Nayak, P. Sriganesh, K. M. Rakshitha, M. V. M. Kumar, B. S. Prashanth, and H. R. Sneha. Literature Review on Phenomenon of Concept Drift and its Handling Approaches. In *2021 Asian Conference on Innovation in Technology (ASIANCON)*, page 7, Piscataway, NJ, USA, 2021.
- [33] B. Obeidat, B. Smith, and B. Heintz. Fine-Grained Time Series Forecasting at Scale with Facebook Prophet and Apache Spark. <https://databricks.com/blog/2020/01/27/time-series-forecasting-prophet-spark.html>, 2020. Accessed: 2022-06-01.
- [34] OECD. Section 4: Guidelines for the reporting of different forms of data. In *Data and Metadata Reporting and Presentation Handbook*. OECD, Paris, 2005.
- [35] N. Oza. Online Bagging and Boosting. In *Conference Proceedings - IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2340–2345, 11 2005.
- [36] T. Ozaki and P. Thomson. A Non-linear Dynamic Model for Multiplicative Seasonal-Trend Decomposition. *Journal of Forecasting*, 21:107–124, March 2002.
- [37] R. Pelossof, M. Jones, I. Vovsha, and C. Rudin. Online Coordinate Boosting. *IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1354–1361, 2009.
- [38] M. D. Pietro. Time Series Analysis for Machine Learning: Trend, Outliers, Stationarity, Seasonality. <https://towardsdatascience.com/time-series-analysis-626bee0d0205>, 2020. Accessed: 2022-06-23.
- [39] D. M. Rocke and D. L. Woodruff. Identification of Outliers in Multivariate Data. *Journal of the American Statistical Association*, 91(435):1047–1061, 1996.
- [40] B. Roy. All about Feature Scaling. <https://towardsdatascience.com/all-about-feature-scaling-bcc0ad75cb35>, 2020. Accessed: 2022-06-01.
- [41] E. Slutsky. The Summation of Random Causes as the Source of Cyclic Processes. *Voprosy Koniunktury*, pages 34–64, 1927.
- [42] S. Tan, K. Ting, and F. T. Liu. Fast Anomaly Detection for Streaming Data. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 1511–1516, January 2011.
- [43] S. C. Tan, K. M. Ting, and F. T. Liu. Fast Anomaly Detection for Streaming Data. In *IJCAI*, pages 1511–1516, 2011.
- [44] S. Vidhyut, S. K. Uppada, and B. SivaSelvan. An Online Supervised Learning Framework for Crime Data. In *Proceedings of the International Conference on Deep Learning, Artificial Intelligence and Robotics (ICDLAIR) 2019*, pages 21–36, Cham, Switzerland, 2021.
- [45] P. Whittle. *Hypothesis Testing in Time Series Analysis*. Statistics, Uppsala Universitet. Almqvist & Wiksells, 1951.

- [46] Wikipedia. Froth flotation. https://en.wikipedia.org/wiki/Froth_flotation, 2022. Accessed 2022-06-15.
- [47] K. Wu, K. Zhang, W. Fan, A. Edwards, and P. Yu. RS-Forest: A Rapid Density Estimator for Streaming Anomaly Detection. In *IEEE International Conference on Data Mining*, volume 2014, December 2014.
- [48] C. Xueqi, J. Xiaolong, Y. Jing, and X. Jun. Technological progress and trends of big data. *Science & Technology Review*, 34(14):49–59, July 2016.
- [49] M. M. W. Yan. Accurate detecting concept drift in evolving data streams. *ICT Express*, 6(4):332–338, 2020.
- [50] S. F. Yilmaz and S. S. Kozat. Pysad: A Streaming Anomaly Detection Framework in Python. *arXiv*, September 2020.
- [51] G. U. Yule. Why Do We Sometimes Get Nonsense-Correlations between Time Series? A Study in Sampling and the Nature of Time Series. *Journal of Royal Statistical Society*, pages 1–64, 1926.
- [52] J. Zenisek, F. Holzinger, and M. Affenzeller. Machine learning based concept drift detection for predictive maintenance. *Computers & Industrial Engineering*, 137:53–65, November 2019.