



M.Sc. THESIS

Identification of a Drinking Water Softening Model using Machine Learning

J.N. Jenden
January 2020

Faculty of Electrical Engineering, Mathematics and
Computer Science (EEMCS)

Graduation committee:

Prof.dr. H.J. Zwart (UT)
Dr. C. Brune (UT)
Prof.dr. A.A. Stoorvogel (UT)
Ir. E.H.B. Visser (Witteveen+Bos)

Company:

Witteveen+Bos
Deventer, the Netherlands

Control Theory
Department of Systems and Control
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands



UNIVERSITY OF TWENTE.

Contents

<i>Table of contents</i>	ii
<i>List of figures</i>	iii
<i>Abstract</i>	iv
<i>Acknowledgements</i>	v
<i>Acronyms and Term Dictionary</i>	vi
<i>Chapter 1: Introduction</i>	1
Softening Treatment Process	1
Hard Water	1
What is Machine Learning?	2
Train, Validation and Test Data	2
Previous Research	2
Aim of the Report	3
Report Layout	3
<i>Chapter 2: Background Information</i>	6
Softening Process Configuration	6
Pellet Softening Reactor	6
Control Actions in the Softening Treatment Step	7
pH as a Control Variable	8
<i>Chapter 3: Data Pre-Processing and Data Analysis</i>	9
Time Series	9
Normalising the Data	9
Removing Corrupted Data	9
Pearson's Correlation Coefficient	10
Autocorrelation	10
<i>Chapter 4: Machine Learning</i>	11
Supervised Learning	11
Train-Validation-Test Data Splitting Method	11
Walk-forward Data Splitting Method	12
Hyperparameters and Hyperparameter Grid Searches	12
Overfitting and Underfitting	13
Evaluation Metrics	13
<i>Chapter 5: Neural Networks and XGBoost</i>	15
Neural Networks	15
Recurrent Neural Networks (RNNs)	15
Memory Cells	16
Standard Time Series NN Structure	17
LSTM Cells	17
Regularisation using a Dropout Layer	18
Gradient Descent	19

Introduction to Decision Trees	20
Difference between Classification and Regression Trees	20
Introduction to XGBoost	20
Feature Importance	21
XGBoost and RNN Model Prediction Horizons	21
<i>Chapter 6: Methods</i>	22
Identification of Inputs and Outputs	22
Data Collection	23
Data Pre-Processing and Data Analysis	23
Prediction	29
<i>Chapter 7: Machine Learning Results</i>	30
RNN Train-Validation-Test Model	30
RNN Walk-Forward Models	31
XGBoost Train-Validation-Test Model	35
XGBoost Walk-Forward Models	36
<i>Chapter 8: Discussion and Conclusions</i>	37
<i>Chapter 9: Recommendations</i>	40
<i>Bibliography</i>	42
<i>Appendix A: Drinking WTP Example and Softening Process Background Information</i>	44
Example WTP	44
Water Flux	44
Water Hardness Chemistry	45
Bypass	45
Calcium Carbonate Crystallisation Reaction	46
<i>Appendix B: Data Analysis</i>	47
Pearson's Correlation Coefficient Matrix	47
Box Plot	47
<i>Appendix C: Machine Learning</i>	50
Python vs Matlab®: Machine Learning and Control Theory Implementation	50
RMSProp Optimisation	50
Derivation of Backpropagation Equations	51
The Backpropagation Algorithm	53
Logistic Activation Function	54
<i>Appendix D: eXtreme Gradient Boost (XGBoost)</i>	55
Regularisation Learning Objective	55
Gradient Tree Boosting	56
<i>Appendix E: XGBoost and RNN Implementation</i>	57
XGBoost Results	57
RNN Results (F=24[hr])	61
XGBoost Hyperparameter Selection	63
RNN Hyperparameter Selection	64
RNN Walk-Forward Training (F=1 [min])	64
RNN Train-Validation-Test Training (F=1 [min])	65
RNN Walk-Forward Training (F=24 [hr])	66
RNN Walk-Forward Training (F=4 [hr])	68
XGBoost Train-Validation-Test Training	69
XGBoost Walk-Forward Training	70

List of Figures

1	Softening treatment process diagram	1
2	Water Treatment Plant (WTP) standard set-up	6
3	Typical pellet softening fluidised bed reactor	7
4	Supervised learning example	11
5	Train-validation-test data split	12
6	Walk-forward	12
7	Simple network	15
8	Recurrent Neural Network (RNN) over time	15
9	Memory cell	16
10	Example RNN structure	17
11	LSTM	17
12	Dropout layer	18
13	Gradient descent	19
14	Regression tree example	20
15	Gradient boosting	21
16	XGBoost and RNN model features and targets	21
17	Data interpolation method	23
18	Caustic soda dosage flow rate and flow rate	24
19	pH autocorrelation	26
20	Mean pH over one day	27
21	pH box plot over hours.	27
22	A month of data from a drinking water reactor	28
24	RNN train-validation-test model prediction	30
25	The first twp RNN walk-forward models	31
26	The last four RNN walk-forward models	32
28	Walk-forward forecasting RNN model predictions	34
29	XGBoost train-validation-test split model prediction	35
30	Feature Importance Train-Validation-Test XGBoost Model	35
31	Weesperkarspel Water Treatment Plant (WTP)	44
32	Water flux	45
33	Boxplot	47
34	Hourly mean of data features and the seeding and draining per hour	49
35	Logistic Activation Function	54
36	Tree ensemble model example	55
37	First four XGBoost walk-forward split model predictions	58
38	Last three XGBoost walk-forward model predictions	59
39	Feature importance of XGBoost models 0 to 3	60
40	Feature importance of XGBoost models 4, 5 and 6	61
41	Walk-forward forecasting RNN model predictions	62

Abstract

This report identifies Machine Learning (ML) models of the water softening treatment process in a Water Treatment Plant (WTP), using two different ML algorithms applied on time series data: eXtreme Gradient Boost (XGBoost) and Recurrent Neural Networks (RNNs). In addition, a control method for the draining of pellets in the softening reactor is explored based on collected softening treatment data and the resulting ML models. In particular, the pH is identified as a potential variable for the control of pellet draining within a softening reactor. The pH forecasts produced by ML models are able to predict the future behaviour of the pH and potentially anticipate when the pellets should be drained.

For implementation of the ML algorithms, the inputs and outputs of the ML models are first identified. Wherein, the pH within the softening reactor is selected as the output, due to its potential control properties. Subsequently, water softening treatment data is collected from a water company residing in the Netherlands. After collection, the data is pre-processed and analysed to be able to better interpret the ML results and to improve the performance of the ML models trained. During pre-processing, the implementation of two ML data splitting methods, walk-forward and train-validation-test, is carried out. The performance of the models is gauged using two different evaluation metrics: Mean Squared Error (MSE) and R-squared. Lastly, predictions are carried out using the trained ML models for a set of forecast horizon lengths.

Comparing the XGBoost and RNN pH predictions, the RNN performs in general better than the XGBoost method, where the RNN model with a train-validation-test split, has a MSE value of 0.0004 (4 d.p.) and an R-squared value of 0.9007 (4 d.p.). Extending the forecast horizon to four hours for the RNN walk-forward model yielded MSE values below 0.01, but only negative R-squared values. Thereby, suggesting that the prediction is relatively close to the actual data points, but does not follow the shape of the actual data points well.

The evaluation metric results suggest that it is possible to create a good performing model using the RNN method for a forecast horizon length equal to one minute. Alternatively, this model is heavily dependent on the current pH value and therefore is deemed to be not a good predictor of the pH. Increasing the horizon length leads to only slightly lower MSE values, but the R-squared values are in general negative, indicating a poor fit.

Keywords: Machine Learning (ML), water softening treatment, Water Treatment Plant (WTP), time series, eXtreme Gradient Boost (XGBoost), Recurrent Neural Network (RNN), pH, control, pellet draining, softening reactor, forecast, inputs, outputs, pre-process, data splitting method, walk-forward, train-validation-test, evaluation metric, Mean Squared Error (MSE), R-squared, prediction, forecast horizon

Acknowledgements

This thesis report is the final product of a team effort. I would like to express my gratitude to a number of people that helped me reach this final stage of publication.

Firstly and most importantly, I would like to thank ir. Erwin Visser, my supervisor at Witteveen+Bos (W+B), for formulating such an interesting topic for my thesis and allowing me to carry out my research at W+B. Furthermore, his technical insights on the water softening treatment system were incredibly useful and helped me better understand the system during our weekly meetings. Additionally, I would like to thank the whole of the Process Automation team at W+B for their support during my entire thesis. Particularly Ko Roebbers and Eddy Janssen, for bringing me in contact with a water company.

Secondly, I would like to send a special thanks to Prof. Hans Zwart, my supervisor from the University of Twente, who gave me technical nuggets of information during my research and helped me shape my final report.

Next, I would like to thank my fellow Systems and Control class mate Anne Steenbeek for his wisdom and advice about the machine learning implementation part of my research. In addition, I want to thank Akhyar Sadad for explaining his machine learning implementation in Python.

Finally, I would like to thank my colleagues Eleftheria Chiou and Shanti Bruyn for their technical input.

Acronyms and Term Dictionary

Acronym	
CPU	Central Processing Unit
IQR	Interquartile Range
LSTM	Long Short-Term Memory
ML	Machine Learning
MPC	Model Predictive Control
MSE	Mean Squared Error
NN	Neural Network
PHREEQC	pH-Redox-Equilibrium Calculations
RAM	Random Access Memory
RMSProp	Root Mean Square Propagation
RNN	Recurrent Neural Network
WTP	Water Treatment Plant
WWTP	Wastewater Treatment Plant
XGBoost	eXtreme Gradient Boost

Term	Definition
Activation function	a function, that transforms the summed weighted input from the neuron into the output
Backend	the computing task that is performed in the background. The user is not able to observe this task being carried out
Backpropagation	an algorithm used during the training of a Recurrent Neural Network (RNN) model
Break through	the act of the grains in the pellet softening reactor being flushed out of the softening reactor to the following stage of the Water Treatment Plant (WTP)
Corrupted data	data containing blanks, NaNs, null-values or other placeholders
Crystallisation rate	the rate at which a solid forms, where the atoms are organised into a crystal structure
Dissolution	the action of becoming incorporated into a liquid, resulting in a solution
Effluent	the water exiting a softening treatment reactor
Ensemble Learning	building a model based on an array of other models [6]
Eutrophication	the phenomenon of an excessive richness of nutrients in a body of water, causing a surge of plant growth
Feature	a measurable property of a system. Although, the <i>target</i> of the machine learning algorithm is not considered a feature
Frontend	the graphical user interface provided to the user when operating software
Horizon span	the array of forecast time steps of a particular model
Hyperparameter	a parameter of a learning algorithm and external to the model
Influent	the water entering a softening treatment reactor
Ion	an atom or a molecule that possesses a positive or negative charge
Learner variable	a variable that is used during the training of a machine learning model
Linear regression	the calculation of a function that minimises the distance between the fitted line and all of the data points. The line is often referred to as the <i>regression line</i>
Lookback	the number of past time steps used to make a model prediction
Misclassify	the act of a result being wrongly classified
Model performance indicator	a statistical measure of the performance of the model against the test set. This term is also often referred to as an evaluation metric
Model Predictive Control (MPC)	a method of system control, which seeks to control a process while satisfying a set of constraints
Overfitting	learning the detail and noise of the training data to the extent that it negatively impacts the performance of the model on new data [12]
Predictor	a variable employed as an input in order to determine the target numeric value in the data
Redox reaction	a reaction where a transfer of electrons is involved
Regularisation	the process of adding information in order to prevent <i>overfitting</i>
Response variable	the target (output) of a decision tree
Supervised learning	the process of feeding a machine learning algorithm with example input-output training data pairs
Target	the output for a machine learning model
Water treatment	act or process of making water more useful or potable [20]
Window of data	a number of rows extracted from the original dataset

Chapter 1: Introduction

1.1. Water Treatment Plant (WTP)

The purpose of a Water Treatment Plant (WTP) is to remove particles and organisms that cause diseases, protect the public's welfare and provide clean drinkable water for the environment, people and living organisms [20]. As of 2016, there are ten different drinking water companies in the Netherlands, employing 4,856 workers. A total 1,095 million m³ of drinking water was produced in the Netherlands in 2016 [23]. An overview of an example WTP can be found in Appendix A.

1.1.1. Softening Treatment Process

Once the water has been pre-treated, it undergoes softening in the WTP. A popular softening process setup is displayed in Figure 1. The raw water enters the process and flows through to the pellet-softening reactor. The softened water then exits the reactor at the top and is subsequently mixed with the bypassed water. Finally, the water is dosed with a form of acid to ensure that the pH reduces. A large pH value kills the bacteria in the downstream biofilters. In addition, the acid counteracts the chemical reactions involving caustic soda. These reactions can potentially negatively impact the downstream WTP equipment. The bypass (described in Appendix A) and raw water flow are controlled using valves.

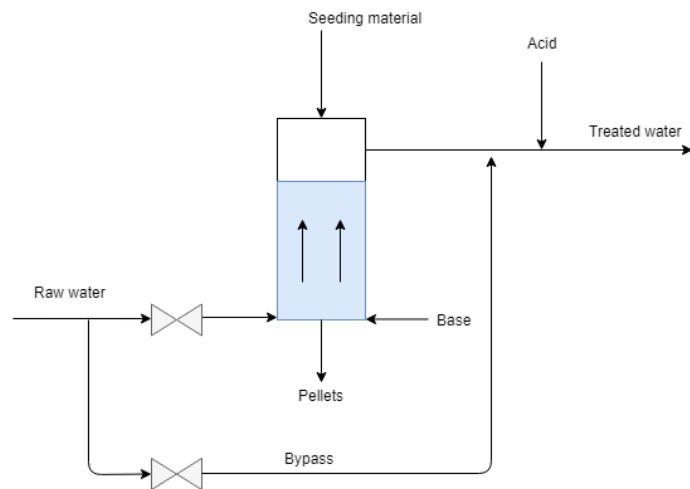


Figure 1: Softening treatment process diagram.

1.1.2. Hard Water

Magnesium and calcium ions are dissolved when water comes into contact with rocks and minerals. The hardness of the water is the total amount of dissolved metal ions in the water. In practice, the hardness is determined by adding the concentration of calcium and magnesium ions in the water, since they are generally the most abundant in the *influent*. Hard water can cause the following problems [1]:

- Decreasing the calcium concentration in water gives rise to a higher pH value in the distributed water, leading to a decrease in the *dissolution* of copper and lead in the distribution system. Ingestion of large quantities of dissolved copper and lead has negative effects on the public's health.
- A higher detergent dosage for washing is required for harder water. This increases the concentration of phosphate in wastewater and contributing to the *eutrophication* effect. Furthermore, a greater usage of detergent increases the average household costs.
- Hard water causes scale buildup in heating equipment and appliances, causing an increase in energy consumption and equipment defects.
- Hard water tastes worse than soft water.
- The damaging or staining of clothing during a wash is often caused by hard water.

1.2. Machine Learning

1.2.1. What is Machine Learning?

Machine Learning (ML) is the science of programming computers using algorithms, so they can *learn from data* [6]. The algorithms develop models, which are able to perform a specific task, relying only on inference and patterns. A more formal definition is as follows:

A computer program is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E .

ML has been frequently used to solve various real-life problems in recent years. One example is predicting the stock market, where stakeholders frequently want to predict future trends of the stock prices. Implementing a ML algorithm using past stock data allows you to generate a model that can predict the future trajectory of the stock price.

1.2.2. Train, Validation and Test Data

A dataset is often for ML partitioned into: *train*, *validation* and *test* datasets. The train partition is used for training during the implementation of the ML algorithm. The validation data is used to evaluate the model during training and allows you to effectively tune the *hyperparameters*. A hyperparameter is a parameter of a learning algorithm and external to the model. Therefore, hyperparameters remain constant during the training of a ML model. Checking the model against the validation data allows you to identify if the model is *overfitting* (explained in Section 4.6) on the training dataset. The test partition consists of the data used to determine the final performance of the created model.

1.3. Previous Research

In K.M. van Schagen's research paper, *Model-Based Control of Drinking-Water Treatment Plants* (published in 2009) [4], the softening process in the Weesperkarspel WTP is evaluated. K.M. van Schagen proposes controlling the pellet-bed height using a Model Prediction Controller (MPC). The MPC determines the seeding material dosage and pellet discharge, to maintain the optimal pellet diameter and maximum bed height under variable flow in the reactor and corresponding temperature.

Stimela is a modelling environment for drinking water treatment processes (including the water softening process) in Matlab®/Simulink® [24]. Stimela was developed by DHV Water BV and Delft University of Technology. The Stimela models calculate changes of water quality parameters, such as pH, pellet diameter and pellet-bed height.

PHREEQC (pH-Redox-Equilibrium Calculations) is a model that was developed by Uninited States Geological Survey (USGS) to calculate the groundwater chemistry [10]. PHREEQC is comprised of all the relevant chemical balance equations for water chemistry, such as acid-base and *redox* (reduction/oxidation) reactions. The PHREEQC (in combination with a model of the calcium carbonate crystallisation rate) simulates a pellet softening reactor [5].

A. Sadad published a research paper in 2019 [9], about a general step-by-step method of applying ML analysis on a time-series dataset. One example case featuring in the research is a Wastewater Treatment Plant (WWTP) system, where an accurate ML model was developed using Recurrent Neural Networks (RNNs) and the XGBoost algorithm (see Chapter 5). Both of the ML algorithms were implemented in Python. In this research, an adapted version of A. Sadad's implementation is employed. This research

firstly seeks to verify the step-by-step method proposed by A. Sadad and secondly, to adapt the implementation to be able to forecast further into the future.

There are no research publications about applying ML on the drinking water treatment softening process. One purpose of this research, is to investigate if it is possible to apply ML on a drinking water treatment softening process and generate an accurate model of the process.

1.4. Aim of the Report

The aim of the research is as follows:

Develop a control strategy that efficiently controls the seeding and draining of the softening reactor based on the pH, using a model developed through ML.

Moreover, this research seeks to answer the following questions:

1. Is it possible to develop a model of the softening treatment process of a WTP using ML?
2. Is the data provided for this research sufficient to develop a model using ML?
3. Can the seeding and draining control be improved by developing a control strategy using the produced ML model?

1.5. Report Layout

The report is organised as follows:

- **Chapter 1. Introduction:** In this chapter, an introduction to the problem and background information about ML and WTPs are given. In particular, this chapter briefly describes: the softening treatment process within a WTP, the problems associated with hard water, a definition of ML and the different partitions of the dataset used for ML implementation. Finally, the relevant previous research is identified and an aim of the report is defined.
- **Chapter 2. Background Information:** In this chapter, further background information about the softening process in a WTP is presented. Beginning with a description of a typical softening process configuration, explaining the role of reserve softening reactors in the softening process. Next, the main features of a pellet softening reactor are described, including the standard WTP pellet discharge action. Furthermore, the key control actions that take place in the softening process are described, including caustic soda dosing in the softening reactor. Lastly, the properties of the pH are described and its potential for use as a draining control variable in the softening process is explained.
- **Chapter 3. Data Pre-Processing and Data Analysis:** Firstly, this chapter describes time series data, the *Z-score normalisation method* and the applicability of normalising the data used for ML. Afterwards, methods of removing erroneous values from the dataset are explored, where erroneous rows and columns can be removed or interpolation applied. Finally, *Pearson's correlation coefficient* and the *autocorrelation* are mathematically described.
- **Chapter 4. Machine Learning:** In this chapter, ML principles are described, where the concept of supervised learning is introduced and two techniques used to split data before applying ML algorithms are introduced: *walk-forward* and *train-validation-test* data splitting. Next, the role of hyperparameters is explained, along with examples of hyperparameters featuring in a Neural Network (NN) and the use of implementing a hyperparameter grid search. Lastly, two so-called *evaluation metrics* are introduced with an explanation of how they can be interpreted.

- **Chapter 5. Neural Networks and XGBoost:** In this chapter, the ML theory is studied in more depth by exploring two prominent ML algorithms used for time series problems: NNs and eXtreme Gradient Boost (XGBoost). Firstly, the NNs section introduces the main components of a NN and how the NN parameters update during training, using samples from a dataset via the gradient descent algorithm. Subsequently, the notion of a RNN is introduced, where past outputs are incorporated into the NN. This concept is expanded on, by describing the function of a Long Short-Term Memory (LSTM) cell, where past outputs are selectively retained based on a mathematical algorithm. Thereafter, the purpose of regularisation is explained and a pertinent NN regularisation technique (the *dropout layer*) is described. Then, a typical RNN structure is described. In the beginning of the XGBoost section: decision trees are introduced, a distinction between classification and regression trees is shown and a relevant example is depicted. Afterwards, the XGBoost algorithm is summarised with its associated *feature importance score*. The chapter is brought to a conclusion, by comparing the *prediction horizons* of the XGBoost and RNN models, giving an indication of their predictive qualities.
- **Chapter 6. Methods:** In this chapter, the methods used to apply the ML algorithms featured in Chapter 5 are explained, which leads to the results shown in Chapter 7. Firstly, the inputs and outputs of the proposed model are identified, using knowledge of the softening process introduced in Chapters 1 and 2. Thereafter, the data is collected from the given water company, taking into account data interpolation and deciding upon a suitable data time interval. Next, the delivered data is pre-processed and analysed using techniques described in Chapters 3 and 4, as preparation for the ML algorithms. Finally, the ML prediction phase methods are explained, making use of the theory of the ML algorithms introduced in Chapter 5. Including an explanation of the hyperparameter selection for the ML algorithms.
- **Chapter 7. Machine Learning Results:** In this chapter, the ML results generated using the methods from Chapter 6 are analysed. In particular, the evaluation metrics described in Chapter 4 are used to assess the performance of the different generated models.
- **Chapter 8. Discussion and Conclusions:** In this chapter, the results of Chapter 7 are discussed and conclusions are drawn based on the results. The conclusion seeks to answer the questions posed in the Aim of the Report Section (Section 1.4).
- **Chapter 9. Recommendations:** In this chapter, recommendations are given for further analysis, including tips for improving the performance of a model generated using the ML algorithms and the practical implications associated.
- **Appendices:** The appendices provide supplementary information to the reader. Appendix A describes an example WTP, as well as the pre-treatment process. In addition, the description outlines where the softening process is positioned in the softening treatment process. In the remainder of Appendix A, the dynamics of water flux in the softening reactor, water hardness chemistry, bypass component in the softening treatment process and calcium carbonate crystallisation reaction are explained. In Appendix B, the *Pearson's correlation coefficient matrix* for the dataset used in this research is shown and a brief description of the main features of a *box plot* are given. Moreover, a figure of the hourly mean of the variables in the dataset is displayed. In Appendix C, the pros and cons of using Python and Matlab® for ML and control theory implementation are considered. Furthermore, a modified version of the *gradient descent algorithm (RMSProp Optimisation)* is considered, along with a detailed derivation of the *backpropagation algorithm*, a summary of the steps taken in the backpropagation algorithm and the *logistic activation function*. In Appendix D, the XGBoost ML algorithm is described. In Appendix E, additional results of the XGBoost and

RNN algorithms are depicted. In addition, the hyperparameter choices for each ML algorithm are described and the Python training logs are given.

Chapter 2: Background Information

2.1. Introduction

In this chapter, the softening process in a Water Treatment Plant (WTP) is described in greater detail. An example softening process configuration is introduced, demonstrating the function of reserve softening reactors. A typical pellet softening reactor and the general softening processes are described, such as the draining of the pellets. Information is provided about the control actions and behaviour present in the process, which seeks to aid the analysis of the dataset and provide more insight into potential control strategy improvements. Finally, the properties of the pH are explained, along with reasoning as to why the pH could be used as a control variable within the system.

2.2. Softening Process Configuration

A typical WTP reactor configuration is shown in Figure 2. In this example the reactors are split into groups of three, consisting of two active reactors (shown in green) and one reserve reactor (shown in orange). The active reactors are consistently used in the process, unless the reactor needs to be switched off. For instance, a reactor may need to be unclogged or components within the reactor replaced. Once an active reactor is switched off, the influent water is redirected to the reserve reactor, thereby giving continuity to the process. The reserve reactors also give flexibility to changes in *effluent* demand. If the effluent demand increases, the reserve reactors are switched on, thus increasing the softening capacity. Simultaneously, the influent flow is increased by pumping more water from the raw water collection points. Having multiple groups of reactors, allows maintenance to be carried out on one group, while the other groups can continue softening the influent.

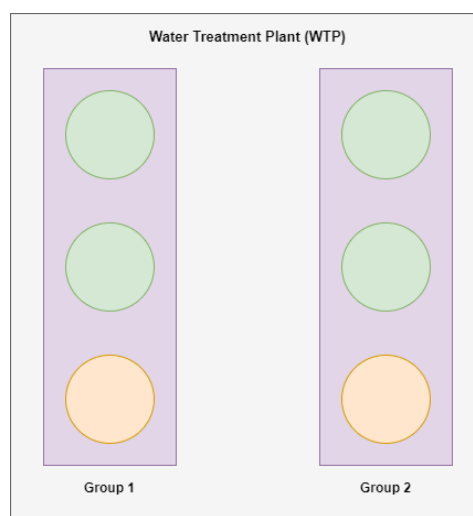


Figure 2: Water Treatment Plant (WTP) standard configuration. The green circles represent the active pellet softening reactors and the orange the reserve reactors.

2.3. Pellet Softening Reactor

The cylindrical pellet softening reactors used at one of the WTPs of Waternet have a diameter of 2.6 meters and a height of 6 meters. These reactors have a capacity of approximately 4800 m³/h [10]. Figure 3 displays an image of a typical pellet softening reactor.

During the pellet softening process, water is pumped in an upward direction in the reactor. The hard water is supplied to the reactor via the pipe labeled A in Figure 3 and the reactor is filled with seeding material. Calcium carbonate *crystallisation* takes place on the surface of the seeding material, leading to a variation of pellet sizes being deposited in layers on the circular plate. More specifically, the heavier larger pellets form the bottom layer of the bed and the smaller pellets accumulate on top. The flow of water through the reactor, causes the majority of the pellets to swirl around above the circular plate in their associated layers. Dosing heads span the width of the circular plate, allowing the supplied water at the bottom of the reactor to pass through. Caustic soda is fed into the reactor via the pipe labeled B. The caustic soda is required for the calcium carbonate crystallisation process that takes place on the surface

of the seeding material. The outgoing water from the reactor (through pipe E) is called the *effluent*.

The presence of the pellets in the reactor results in a pressure difference across the reactor. Pressure difference measurements across the length of the reactor are used to control the automatic pellet discharge [4] (draining is facilitated by tap C shown in Figure 3). When the pellet diameters grow the pressure difference increases. Once this pressure difference exceeds a certain value set by the operators of the given WTP, the pellets are automatically discharged.

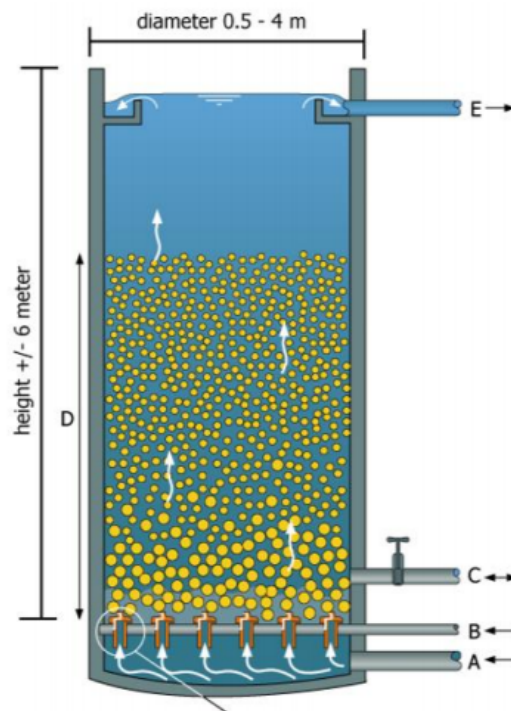


Figure 3: Typical pellet softening fluidised bed reactor [3].

2.4. Control Actions in the Softening Treatment Step

The main control actions in the pellet softening reactor are as follows [4]:

- Water flow through the reactor. This is controlled using a series of pumps upstream from the softening treatment step in the WTP.
- Base dosing (caustic soda is often used). This impacts the pH of the water in the reactor and consequently the rate of crystallisation. A higher base dosing generally leads to a lower hardness and a greater pH.
- Seeding material dosage. One example of a frequently used seeding is sand. Adding a greater mass of seeding material to the reactor, leads to a greater surface area for crystallisation to take place.
- Seeding material diameter. Selecting a seeding material with a smaller diameter grain gives rise to a larger surface area (per kilogram of seeding material). At the same time, the grains need to be heavy enough to prevent them from *breaking through* to the next stage of the WTP.
- Pellet discharge. The pellet discharge action is controlled by the pressure difference across the reactor. An accumulation of large pellets causes the pressure difference to increase. The pressure

difference threshold value can be adjusted by the operator at a certain WTP. Increasing the threshold value leads to a lower discharge rate. Thereby, leading to an increase in the size of the pellets in the pellet-bed and consequently less surface area for crystallisation to occur. Moreover, it can cause blockages in the reactor, due to a decrease in the *porosity* of the pellet-bed. Conversely, decreasing the threshold value increases the frequency of discharges, generally leading to pellets with a lower diameter in the bed. An increase in discharges, requires more seeding material to be added to the reactor, therefore leading to higher softening treatment costs.

2.5. pH as a Control Variable

The pH describes the acidity or alkalinity of a solution and common values range from 0 to 14, where 7 indicates neutrality of the solution (at 25°C). Values less than 7 (at 25°C and a certain salinity) implies an acid solution and greater than 7 (at 25°C and a certain salinity) an alkaline solution. More formally, the pH is the *decadic* logarithm (logarithm with base 10) of the reciprocal of the hydrogen activity in a solution, where the hydrogen ion activity is denoted as a_{H^+} and is described by the following mathematical formula:

$$\text{pH} = -\log_{10}(a_{H^+}) = \log_{10}\left(\frac{1}{a_{H^+}}\right).$$

Once the pellets tend to saturation, less calcium carbonate is able to crystallise onto the pellets. Leading to surplus caustic soda in the reactor and an increase in pH. Consequently, the effluent becomes harder, due to less metal ions being removed from the water. Therefore, an increase in pH gives a good indication of when the pellets should have been drained.

A high pH in the reactor could kill the bacteria in the downstream biofilters, if the acid dosing downstream from the softening reactor is not able to lower the pH sufficiently. The biofilters are required to eliminate dissolved organic compounds in the water. Moreover, reactions involving caustic soda downstream from the reactor could occur, if the pH is too high after the acid dosing step.

Chapter 3: Data Pre-Processing and Data Analysis

3.1. Introduction

In this chapter, techniques of data pre-processing and data analysis are described. This chapter describes in particular: time series data, normalising data, removing corrupted data, Pearson's correlation coefficient and *autocorrelation*. Pre-processing and data analysis are necessary for datasets generated in the water softening process (described in Chapters 1 and 2). After a dataset is pre-processed, Machine Learning (ML) (Chapters 4 and 5) can be more effectively applied.

3.2. Time Series

A time series is a sequence of discrete time data. The stock market prices are an example of a time series, since the numerical prices are recorded for a given time interval. We are going to solely focus on time series data for our research.

3.3. Normalising the Data

Normalising the train data before training your model, ensures that the input data satisfies the scale of the *activation functions* used during ML. An activation function is a function, that transforms the summed weighted input from the neuron into the output. For example, let \mathbf{w} be the weight vector, \mathbf{x} the corresponding input vector, σ the activation function and y the output. The activation function makes the following transformation: $y = \sigma(\mathbf{w}^T \mathbf{x})$. When the generated model provides a prediction, the data is transformed back to the original scale. Normalisation of the data is sometimes not required, depending on the ML algorithm and the scale of the original data [9]. If the variance of the dataset is relatively large, then it is recommended to normalise the data for ML [9].

The Z-score normalisation is a popular method to normalise the data. This method entails transforming the data to have zero mean and unit variance (equal to one). The transformation is mathematically described as follows:

$$\mathbf{Y}_{new} = \frac{\mathbf{Y}_{old} - E[\mathbf{Y}_{old}]}{\sigma},$$

where \mathbf{Y}_{old} denotes the original data vector and $E[\mathbf{Y}] = \frac{1}{N} \sum_{i=1}^N y_i$ is the mean of the original data. N denotes the number of samples and y_i is the sample i of the original data. Using the same notation, the standard deviation σ is described as follows:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - E[\mathbf{Y}_{old}])^2}. \quad (1)$$

3.4. Removing Corrupted Data

Real life datasets often contain erroneous values such as duplicated or missing values which are frequently encoded as blanks, NaNs, null-values, or other placeholders [9]. The erroneous values reduce the performance of ML algorithms. This is often the case for time series data, since the sensors are required to measure regularly for a given time interval. Thus, if a sensor is for instance, temporarily switched-off, damaged or blocked, missing values arise in the dataset.

To minimise the impact of the erroneous values in the dataset, the samples (rows) holding the erroneous value(s) can be deleted. However, removing time series samples can negatively impact ML, since the ML models learn from past data steps with gaps in. For instance, let a time series be described by

$y_t = \alpha y_{t-1} + \beta u_t$, where y is the output, u the input and t the present time step. If time step $t - 1$ is removed from the dataset, then the previous time step $t - 2$ is used instead (if it is not also removed from the dataset), i.e. the equation becomes $y_t = \alpha y_{t-2} + \beta u_t$. Therefore, the value y_{t-1} is skipped, leading to a gap in the information fed into the ML algorithms. In addition, if the number of corrupted samples is relatively large, it is generally better to find a way to save as many samples as possible, because the data could hold important information about the system. Data samples can be saved by replacing the erroneous values with *interpolated* values. Interpolation means estimating data values based on known sequential-data. Another technique, is to set the missing values to a constant number, such as 0 or a large negative value, depending on the dataset. The idea is that the ML algorithm will ignore the irregular values (outliers) when training the model. All of the methods explained should be taken into consideration when cleaning a particular dataset. One method is likely to perform better than the rest for a given dataset and can often be deduced from knowledge about the system.

3.5. Pearson's Correlation Coefficient

The Pearson's correlation coefficient measures the degree of correlation between two variables. The coefficient (denoted ρ) satisfies $-1 \leq \rho \leq 1$, where 1 indicates strong positive correlation and -1 strong negative correlation. If magnitude of the coefficient is relatively low, then the correlation is considered weak between the two variables. A value of 0 indicates that there is no correlation whatsoever. Pearson's correlation coefficient is described by the following formula:

$$\rho_{\mathbf{X},\mathbf{Y}} = \frac{\text{cov}(\mathbf{X},\mathbf{Y})}{\sigma_{\mathbf{X}}\sigma_{\mathbf{Y}}},$$

where $\rho_{\mathbf{X},\mathbf{Y}}$ signifies the Pearson's correlation coefficient between vectors \mathbf{X} and \mathbf{Y} , $\text{cov}(\mathbf{X},\mathbf{Y}) = \frac{1}{N} \sum_{i=1}^N (x_i - E[\mathbf{X}])(y_i - E[\mathbf{Y}])$ is the covariance between \mathbf{X} and \mathbf{Y} . N symbolises the number of samples, the vector pair (\mathbf{X},\mathbf{Y}) takes on values (x_i, y_i) and $E[\mathbf{X}]$ ($E[\mathbf{Y}]$) is the expected value (mean) of \mathbf{X} (\mathbf{Y}). Furthermore, $\sigma_{\mathbf{X}}$ and $\sigma_{\mathbf{Y}}$ represent the standard deviation of \mathbf{X} and \mathbf{Y} respectively and are calculated as described in equation (1).

3.6. Autocorrelation

Autocorrelation is the degree of similarity between a given time series and a lagged version of itself over successive time intervals [18]. It can be likened to calculating the correlation between two different time series, except autocorrelation employs the same time series twice, i.e. a lagged version and an original. The autocorrelation is defined as follows:

$$\rho_{\tau} = \frac{\sum_{t=\tau+1}^{N-\tau} (x_{t-\tau} - E[\mathbf{X}])(x_t - E[\mathbf{X}])}{\sum_{t=1}^N (x_t - E[\mathbf{X}])^2},$$

where τ ($\in \mathbb{N} \setminus \{0\}$) is the lag and x_t is a sample of vector \mathbf{X} . $E[\mathbf{X}]$ the mean of vector \mathbf{X} and N the number of samples of the variable.

Chapter 4: Machine Learning

4.1. Introduction

In this chapter, key basic Machine Learning (ML) principles are explained. These explanations encompass: *overfitting* and *underfitting*, *hyperparameters*, *supervised learning*, two data splitting techniques and ML *evaluation metrics*. The resulting pre-processed data (described in Chapter 3), needs to be partitioned before applying ML algorithms. Afterwards, so-called hyperparameters can be tactically selected for the given ML algorithm. Finally, the resulting ML model requires a performance evaluation using evaluation metrics. In the ML research domain, a *feature* refers to an input of the ML model and the *target* is the output.

4.2. Supervised Learning

Supervised learning is when you feed your ML algorithm with example input-output training data pairs. The ML algorithm then generates a function (model) that is able to map the input to the output, i.e. $Y = f(X)$, where Y is the output, f a function and X the input. On the other hand, *unsupervised learning* is when the example data fed into the algorithm does not include a corresponding output (only input training data), that it can learn from. Therefore, unsupervised learning learns only from the input X and the corresponding output Y is unknown. In this research, only supervised learning algorithms are implemented, because time series forecasting ML models use supervised learning.

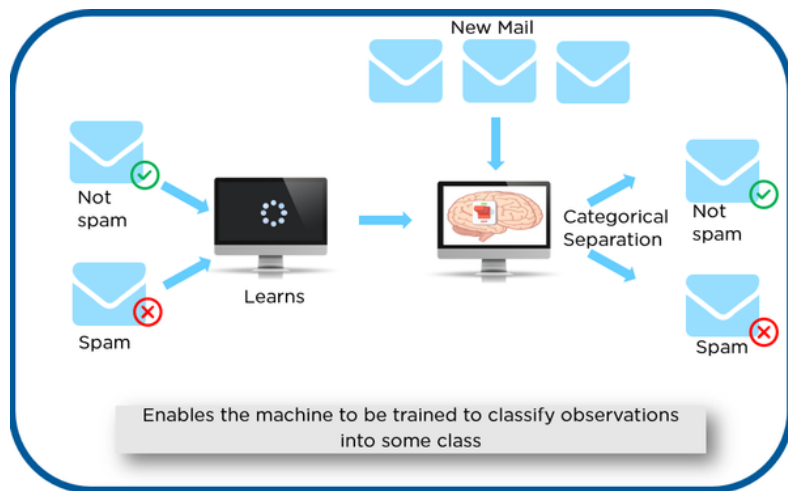


Figure 4: Supervised learning spam identification example. Taken from [26].

An example of supervised learning is illustrated in Figure 4. In this example, the aim is to differentiate between the spam emails and the emails that do not contain spam. The computer is able to learn from previous emails with a corresponding email label, not spam or spam. The spam labels are considered to be the output Y . Implementing ML via the computer, generates the function f and can be subsequently used to classify new emails, where input X consists of the email *features* and the corresponding outputs consist of a prediction of the spam label.

Two possible data splitting methods for supervised ML are: *train-validation-test* and *walk-forward*. The train-validation-test is the most commonly used data splitting method amongst data scientists. The walk-forward method was originally designed by the financial trading industry and is these days frequently applied on a variety of time series datasets.

4.2.1. Train-Validation-Test Data Splitting Method

For the train-validation-test data splitting method, the dataset is partitioned into a *train-validation-test data split* (as illustrated in Figure 5). The *train* partition is used for training during the implementa-

tion of the ML algorithm. The validation data is used to evaluate the model during training and allows you to effectively tune the *hyperparameters* (explained in Section 4.3.). Checking the model against the validation data allows you to identify if the model is *overfitting* on the training dataset. The test partition consists of the data used to determine the final performance of the created ML model.

A data split of 80% train data and 20% test data is often selected as a starting point, where a partition of validation data is not considered a necessity. An adjustment of the data split could be deemed necessary based on the amount of data available. For instance, if there is a large quantity of data available, then a higher percentage can be allocated to the training dataset, since there is considered to be enough test data.

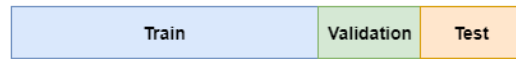


Figure 5: Train-validation-test data split.

4.2.2. Walk-forward Data Splitting Method

The walk-forward validation strategy is used exclusively for time series data analysis. For this strategy, the data is split into *windows*. Each window has the same train-test data split. The train data contains the features (inputs) and target (output) for a given time period. The test data holds the target data (outputs) for a time period following the respective train data time period. The following window is the same length and shifted in time by the length of the test set. This data splitting technique is illustrated in Figure 6. A model is generated for each set of windowed data. The respective model gives a prediction based on the training data and this can be compared against the test data to measure the performance of the model.

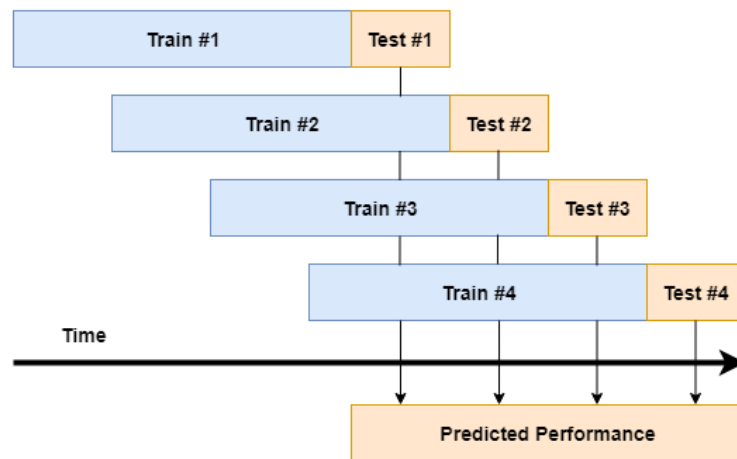


Figure 6: Walk-forward.

Applying the walk-forward validation strategy is useful to validate whether the hyperparameters need to be adjusted to improve the performance of the ML algorithm, since the computation times to generate a model can be considerably lower than the computation times of the train-validation-test method, depending on the length of the window selected. Moreover, the water softening treatment methods can vary in a Water Treatment Plant (WTP) over the course of time, thus the walk-forward validation strategy is able to cope better with these changes by training the given model on only the most recent window of data. For example, the caustic soda dosing method could be altered for a certain WTP.

4.3. Hyperparameters and Hyperparameter Grid Searches

A *hyperparameter* is a parameter of a learning algorithm and external to the model [6]. The hyperparameters are fixed during training of the ML model. A few examples of hyperparameters are: the number of layers in a NN, the amount of neurons in a layer, the type of activation function used in each layer and the learning rate.

The enormous size of potential hyperparameter combinations to train your NN can become overwhelming. To mitigate this problem, it is helpful to use a hyperparameter grid search. This method iterates through a set of hyperparameter combinations and calculates the optimum combination based on evaluation metrics (explained in Section 4.5). Thus, sparing the user from having to manually input new hyperparameters and noting the evaluation metrics at the end of each ML training run. Naturally, there are an infinite number of combinations and the search can only analyse a small portion, due to the computation times.

4.4. Overfitting and Underfitting

Overfitting occurs when a model is generated via Machine Learning (ML) and the resulting model models the training data too closely. In other words, “overfitting happens when a model learns the detail and noise of the training data to the extent that it negatively impacts the performance of the model on new data” (J. Brownlee, 2016) [12]. Thus, the noise or random fluctuations of the training set are learnt as concepts by the model. The resulting model is then not able to generalise as well and therefore is not as effective at dealing with new data.

Overfitting can be reduced by increasing the amount of training data, applying *regularisation* techniques to the ML algorithms or by reducing the size of the neural network.

Underfitting occurs when a model is unable to model the training data nor generalise to new data. A model is said to generalise well to new data, when it is able to make an relatively accurate prediction based on the new data as input. In terms of the two evaluation metrics introduced in Section 4.5, the MSE would be relatively low and the R-squared value close to a positive value of one.

4.5. Evaluation Metrics

Once a model has been created by implementing ML on the training data, a prediction is made using the model. This prediction is then compared against the test data for an indication of model performance. To be able to effectively determine the performance, it is helpful to use a *model evaluation metric*. Two popular evaluation metrics, R-squared and Mean Squared Error (MSE), are described in Subsections 4.5.1. and 4.5.2. respectively.

It is more effective to use multiple indicators in conjunction, since a single indicator is unable to give a full explanation of the model performance, due to each individual indicator having its pros and cons (Krause et al., 2015) [16].

4.5.1. R-squared

R-squared, also known as *coefficient of determination* is a statistical measure of the distance between the data and the *regression predictions*. In other words, the R-squared metric measures the proportion of variance of the actual data points that is described by a model prediction. The mathematical definition is [9]:

$$R^2 \equiv 1 - \frac{SS_{res}}{SS_{tot}},$$

where $SS_{tot} = \sum_i (y_i - E[\mathbf{Y}])^2$ is the total sum of squares (variance multiplied by the number of data points in the dataset) and \mathbf{Y} is the vector of data points y_i (with $i \in \mathbb{N} \setminus \{0\}$). $E[\mathbf{Y}] = \frac{1}{N} \sum_{i=1}^N y_i$ denotes the mean of the dataset, where N ($\in \mathbb{N} \setminus \{0\}$) is the number of data points in the dataset. $SS_{res} = \sum_i (y_i - f_i)^2$ (f_i is a given predicted value) represents the residual sum of squares.

If $R^2 = 1$, the regression prediction fits the actual data points perfectly. On the other hand, $R^2 = 0$ implies that none of the variability of the data points is explained by the prediction around the mean of the data points. Thus, the ultimate aim is to minimise SS_{res} . A value outside the range 0 to 1 occurs when the model fits the data worse than the mean horizontal hyperplane (mean for each dimension). This could indicate that the model is not an appropriate fit for the data.

4.5.2. Mean Squared Error (MSE)

The MSE measures the average of the errors squared, where the error is the difference between the actual data point and the data point generated by the model. As a mathematical function, the MSE is represented as follows:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_i - f_i)^2,$$

where N is the number of predictions, y_i the actual data point at index i ($\in \{1, \dots, N\}$) and f_i the predicted data point at index i .

One criticism of MSE, is that the outliers are heavily weighted. On the other hand, MSE is widely recognised as one of the best error functions.

Chapter 5: Neural Networks and XGBoost

5.1. Introduction

In this chapter, Neural Networks (NNs) and the eXtreme Gradient Boost (XGBoost) algorithm are described. NNs and XGBoost algorithms are often used for Machine Learning (ML) using time series data, since they are able to incorporate relations between past and current time steps in the resulting model. For improved results, the dataset should be pre-processed using techniques described in Chapter 3 and split employing the two splitting techniques in Chapter 4. A resulting NN (or XGBoost) model can be evaluated using the evaluation metrics described in Section 4.6. If the evaluation metrics results are not satisfactory, adjusting the NN (or XGBoost) hyperparameters (described in Chapter 4) can lead to an improved NN (or XGBoost) model. Subsections 5.2.1, 5.2.2, 5.2.3.1 and 5.2.3.2 are largely based on Chapters 11 and 14 from *Hands-On Machine Learning with Scikit-Learn & TensorFlow*, published by A. Géron in 2017 [6].

5.2. Neural Networks

NNs are comprised of layers of neurons with weights interlinking them. A simple NN structure can be observed in Figure 7. The orange circles symbolise the bias neurons, the blue circles represent the hidden neurons, the green circles the input neurons and the purple circles denote the output neurons. The symbol a_1^3 denotes the *activation function* of the first neuron in the third layer. There is one hidden layer in this example. The NN in Figure 7 is a *feedforward network*, since the connections are in a forward direction, i.e. in the direction of the output. The bias (orange circles in Figure 7) neurons are not dependent on previous layers. The purpose of the bias is to create a desired shift in the activation function of a given layer and ultimately generate a better performing model. In this research, a more sophisticated NN, Recurrent Neural Network (RNN), is employed.

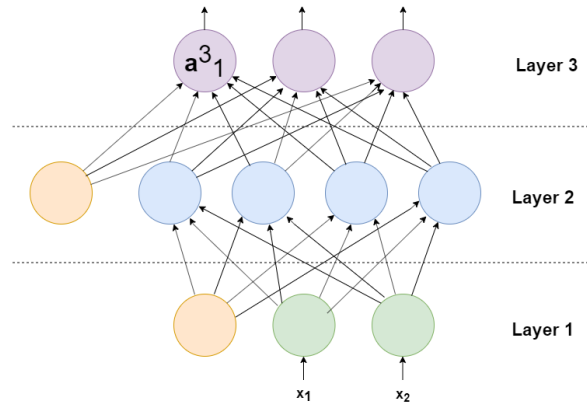


Figure 7: Simple network.

5.2.1. Recurrent Neural Networks (RNNs)

A recurrent network is almost identical to a feedforward network, except it has also connections in a backwards direction. The diagram of a RNN mapped against a time axis can be seen in Figure 8.

For the example feedforward case for a single neuron and a single instance, the output is described as

$$y_{(t)} = \phi(\mathbf{x}_{(t)}^T \cdot \mathbf{w}_x + b),$$

where ϕ represents the given activation function, \mathbf{w}_x the weight for the input \mathbf{x} and b is the bias constant.

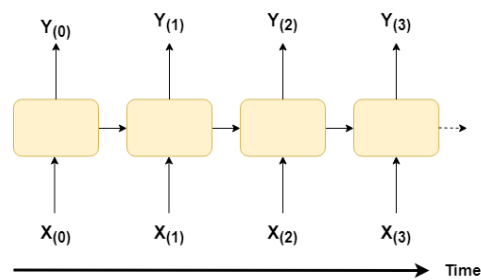


Figure 8: RNN over time.

In comparison the RNN output for a single neuron and single time instance is given by

$$\mathbf{y}_{(t)} = \phi\left(\mathbf{x}_{(t)}^T \cdot \mathbf{w}_x + \mathbf{y}_{(t-1)}^T \cdot \mathbf{w}_y + b\right),$$

where $\mathbf{y}_{(t-1)}$ symbolises the output of the previous time step and \mathbf{w}_y is the corresponding weight. The training data is split into batches and each batch is referred to as a *mini-batch*. This formula can be extended to accommodate multiple recurrent neurons for all instances in a mini-batch, using a vectorised form of the previous equation [6]

$$\mathbf{Y}_{(t)} = \phi\left(\mathbf{X}_{(t)} \cdot \mathbf{W}_x + \mathbf{Y}_{(t-1)} \cdot \mathbf{W}_y + \mathbf{b}\right).$$

- $\mathbf{Y}_{(t)}$ is a $m \times n_{neurons}$ matrix containing the layer's outputs at time step t for each instance in the mini-batch, where m is the number of instances in the mini-batch and $n_{neurons}$ the number of neurons in the layer.
- $\mathbf{X}_{(t)}$ is a $m \times n_{inputs}$ matrix containing the inputs for all instances, where n_{inputs} denotes the number of inputs.
- \mathbf{W}_x is a $n_{inputs} \times n_{neurons}$ matrix containing the connection weights for the inputs of the current time step.
- \mathbf{W}_y is a $n_{neurons} \times n_{neurons}$ matrix containing the connection weights for the outputs of the previous time step.
- \mathbf{b} is a vector of size $n_{neurons}$ containing each neuron's bias term.

Notice that $\mathbf{Y}_{(t)}$ depends on $\mathbf{X}_{(t)}$ and $\mathbf{Y}_{(t-1)}$, which in turn is dependant on $\mathbf{X}_{(t-1)}$ and $\mathbf{Y}_{(t-2)}$, which is dependant on $\mathbf{X}_{(t-2)}$ and $\mathbf{Y}_{(t-3)}$ and so forth. Therefore, $\mathbf{Y}_{(t)}$ is a function of all inputs from time $t = 0$, i.e. $\mathbf{X}_{(0)}, \mathbf{X}_{(1)}, \mathbf{X}_{(2)}, \mathbf{X}_{(3)}, \dots, \mathbf{X}_{(t)}$. At $t = 0$ it is assumed that there are no previous outputs and are taken to be zeros.

5.2.2. Memory Cells

The accumulation of outputs at a recurrent neuron from the previous time steps, can be likened to storing *memories*. A component of a NN that preserves some state across time steps is called a *memory cell*.

Mathematically, a cell's state is represented as $\mathbf{h}_{(t)} = f(\mathbf{h}_{(t-1)}, \mathbf{x}_{(t)})$ [6]. Thus, depending on the input vector of the current time step and the *memory* state of the previous time step. The vector \mathbf{h} stands for "hidden". As a result, the output at time step t , denoted by $\mathbf{y}_{(t)} = z(\mathbf{h}_{(t-1)}, \mathbf{x}_{(t)})$, is a function of the previous memory state and the current inputs. A diagram representation of a memory cell is shown in Figure 9. The left hand side image displays a memory cell. The right hand side shows the pattern of a memory cell over time.

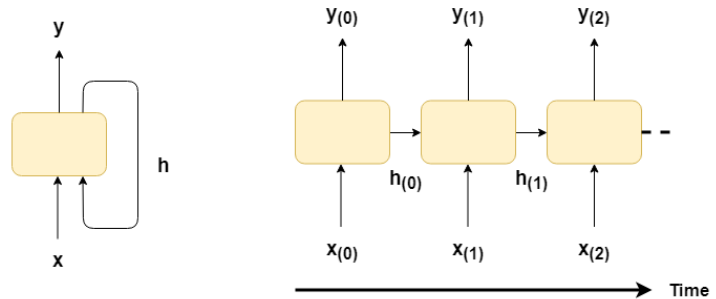


Figure 9: Memory cell.

5.2.3. Standard Time Series RNN Structure

A typical RNN structure for time series data can be viewed in Figure 10, where the general direction is from left to right. The example construction contains two LSTM layers (described in Subsection 5.2.3.1), which are able to draw upon past data-steps. Furthermore, it has a dropout layer (described in Subsection 5.2.3.2), which acts as *regularisation* in the model. It is possible that there exists a better RNN structure for a given time series problem. An improved structure could be found by adjusting the original structure and comparing the evaluation metrics, to determine whether the new model structure performs better. In general, the best time series NNs contain regularisation and LSTM layers.



Figure 10: Example RNN structure.

5.2.3.1. LSTM Cells

The *Long Short-Term Memory* (LSTM) cell was founded in 1997 by Sepp Hochreiter and Jürgen Schmidhuber [21]. The LSTM cell is similar to the memory cell, except that it performs better; training converges faster and it is able to detect long-term dependencies in the data. Training is said to converge faster, when the error plateaus faster during training. A. Géron provides an explanation of the LSTM algorithm and is summarised in the remainder of this Subsection [6].

The architecture of a LSTM cell is displayed in Figure 11. If you ignore the contents of the box, thereby treat it as a black-box, the LSTM cell appears to be a regular memory cell, except the state is split into two vectors: $\mathbf{h}_{(t)}$ and $\mathbf{c}_{(t)}$ (\mathbf{c} denotes "cell"). The $\mathbf{h}_{(t)}$ vector can be considered a short-term (memory) state and $\mathbf{c}_{(t)}$ a long-term state.

Now drawing attention to the contents of the box. The network structure is based on determining what to store in the long-term state, what can be removed and what to read from it. As the long-term state $\mathbf{c}_{(t-1)}$ travels through the network from left to right, it can be observed that it initially goes through a *forget gate*, dropping some information, and subsequently adds some new information via the addition operation (which adds information that were selected by an *input gate*). Finally, the resulting $\mathbf{c}_{(t)}$ exits the box without any further transformations. Furthermore, after the addition operation, the long-term state is replicated and then passes through the tanh function and the output is filtered by the *output gate*. This produces the short-term state $\mathbf{h}_{(t)}$, which is equivalent to the cell's output at the present time step $\mathbf{y}_{(t)}$.

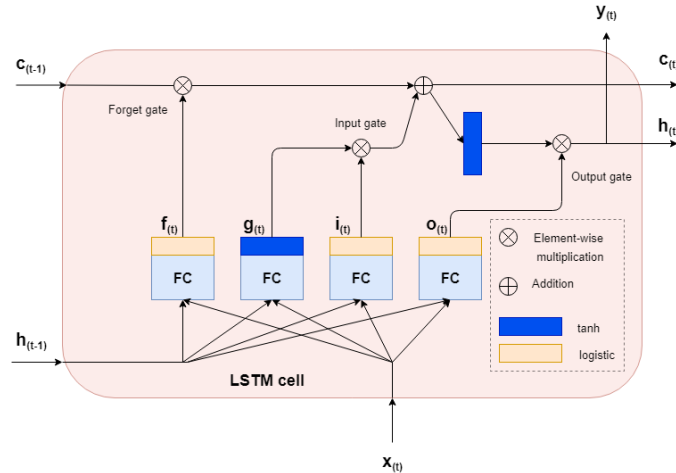


Figure 11: LSTM cell. Note that FC stands for Fully Connected and the definition of the *logistic activation function* is in Appendix C.

The next step is explaining the origin of the new memories and how the gates function. Firstly, current input vector $\mathbf{x}_{(t)}$ and the previous short-term state $\mathbf{h}_{(t-1)}$ are fed to four different fully connected layers. Each fully connected layer has its own function:

- The main layer outputs vector $\mathbf{g}_{(t)}$. It analyses the current inputs $\mathbf{x}_{(t)}$ and the previous short-term state $\mathbf{h}_{(t-1)}$. In a standard memory cell (as described in Subsection 5.2.2), there exist no other neuron layers and its output goes straight out to $\mathbf{y}_{(t)}$ and $\mathbf{h}_{(t)}$. In contrast, in an LSTM cell this layer's output is instead partly stored in the long-term state.
- The remaining three neuron layers are so-called *gate controllers*. Their outputs range from 0 to 1, since they make use of the *logistic activation function* (see Appendix C for the definition). Notice that their outputs are fed to element-wise multiplication operations. Therefore, if they output 0s, the gate is closed and 1s as output, opens the gate. In more detail:
 - The *forget gate* (controlled by $\mathbf{f}_{(t)}$) controls which time steps of the long-term state should be removed.
 - The *input gate* (controlled by $\mathbf{i}_{(t)}$) controls which time steps of $\mathbf{g}_{(t)}$ should be added to the long-term state.
 - The *output gate* (controlled by $\mathbf{o}_{(t)}$) controls which time steps of the long-term state should be read and added to the output at the current time step (for both $\mathbf{y}_{(t)}$ and $\mathbf{h}_{(t)}$).

In summary, a LSTM cell is able to learn to recognise an important input (that is the role of the input gate), store it in the long-term state, learn to preserve it for as long as it is necessary (that is the role of the forget gate) and learn to extract it whenever it is required. This explains why LSTM are very successful in capturing long-term patterns in time series.

5.2.3.2. Regularisation using a Dropout Layer

Regularisation is implemented to reduce overfitting. A frequently used regularisation technique for deep (many layered) NNs is *dropout*. It was proposed by G. E. Hinton in 2012 and subsequently a paper was published giving greater detail by Nitish Srivastava et al. in 2014 [22].

At every training step, every neuron (including the input neurons, but excluding the output neurons) has probability p of being briefly "dropped out". In other words, it will be completely ignored during the current training step, but has the potential to be active during the next step. This algorithm is shown in Figure 12. Note that the green circles represent the input neurons, the blue circles represent the hidden layer neurons and the orange circles symbolise the bias neurons. A cross in the neurons indicates that the neuron is not active for that time step. The hyperparameter p is called the *dropout rate* and is usually set to 50%. The neurons are not dropped once the training is finished. The purpose of this method is to reduce the co-dependencies in the network,

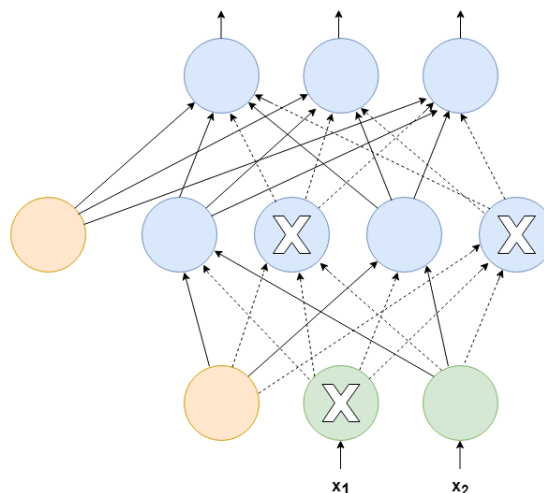


Figure 12: Dropout layer.

thus reducing overfitting.

5.2.4. Gradient Descent

The gradient descent algorithm is a frequently used method to update the weights of the network. Generally, it is an iterative optimisation algorithm for finding the minimum of a function. The algorithm adjusts the weight with a step proportional to the negative gradient of the cost function at a given iteration. A diagram of a visual representation of gradient descent is depicted in Figure 13. The vector of the network weights is updated using the following formula:

$$\mathbf{w}^{(i)} = \mathbf{w}^{(i-1)} - \eta \nabla C(\mathbf{w}^{(i-1)}), \quad (2)$$

where i is the current time step and η is the learning rate hyperparameter. The $\nabla C(\mathbf{w}^{(i-1)})$ part of the second term can be determined using *backpropagation* (see Appendix Section C.3. for more information). A faster gradient optimiser is RMSProp and is explained in Appendix Section C.2.

There are two different algorithms that can be used to apply Gradient Descent on our training data: *Stochastic Gradient Descent* and *Batch Gradient Descent*. The Stochastic Gradient Descent algorithm uses equation (2) to update the weights of the network and the gradient for every training sample. On the other hand, the Batch Gradient Descent algorithm updates the weights only when all the training samples have been fed into the network, therefore using formula (2) only once.

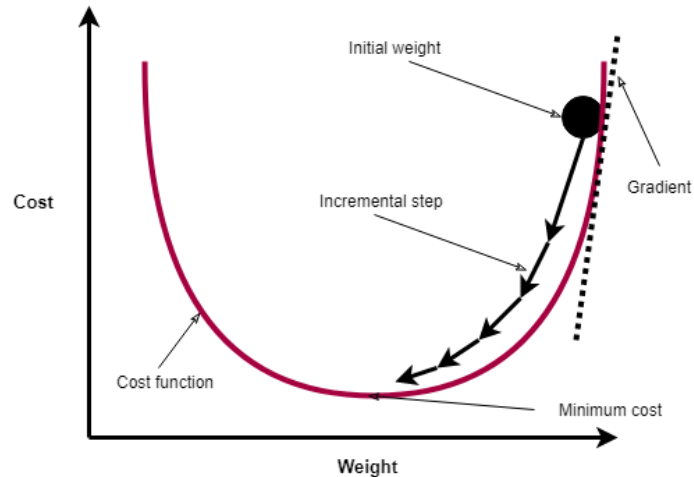


Figure 13: Gradient descent.

5.3. XGBoost

5.3.1. Introduction to Decision Trees

A decision tree is defined by R. S. Brid (2018) as follows [15]:

A *decision tree* is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

5.3.2. Difference between Classification and Regression Trees

The *classification tree* splits the *response variable* (target variable) into discrete values (e.g. 0 or 1) or verbal classes (e.g. *Yes* or *No*). These response variables are frequently referred to as *categorical*. Conversely, the *regression tree* response variable is continuous or numeric and not categorical [13]. We focus only on constructing regression trees during our research, since we wish to forecast numerical data. An example of a simple regression tree can be found in Figure 14 based on typical Water Treatment Plant (WTP) softening data variables. The *root node* in this case is the flow variable and is the beginning point of the algorithm. An example for one sample being processed by the tree is as follows: the flow is higher than Z_1 , so the sample is directed to the right hand side *branch* of the tree. The sample's pellet-bed value is larger than Z_2 , thus the pH is equal to x_3 .

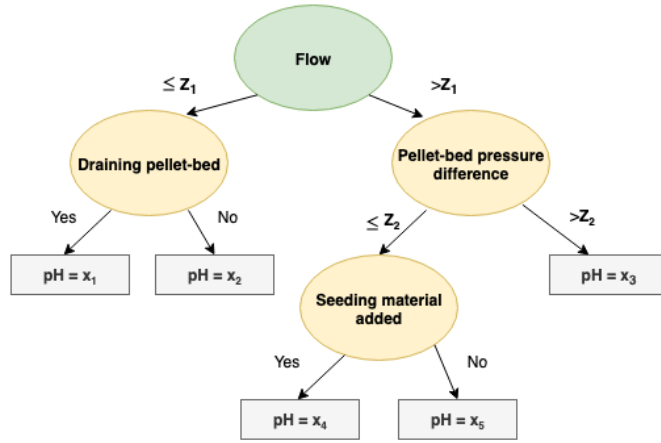


Figure 14: Regression tree example based on WTP softening variables. Note that $x_1, x_2, x_3, x_4, x_5, Z_1, Z_2 \in \mathbb{R}$.

5.3.3. Introduction to XGBoost

In recent years XGBoost was often implemented and has won many ML competitions in Kaggle (online community of data scientists and machine learners), due to its computational speed and model performance. T. Chen introduced the XGBoost technique when she published her work in 2016 [14]. XGBoost stands for eXtreme Gradient Boosting. The XGBoost algorithm builds on the additive optimisation technique called *gradient boosting* by adding a regularisation term to the algorithm to combat overfitting. It is argued that XGBoost is computationally ten times less expensive than the original gradient boosting algorithm on a given machine [9].

The idea of gradient boosting is to convert the *weak learner variables* into *strong learner variables*. This is carried out using the concept *ensemble learning*, where a decision tree model is constructed on the basis of many other existing models. We call a combined model consisting of the best performing models an *ensemble*. An ensemble will perform better than the best individual model. By continually incorporating new models into the ensemble, the error usually reduces on the training set. A visual example of the algorithm iterations can be seen in Figure 15. The points not lying on the plot are considered to be *misclassified*, i.e. the wrong value has been assigned by the model. The XGBoost method is

described in more detail in Appendix D.

5.3.4. Feature Importance

During implementation of the XGBoost algorithm, it is relatively easy to derive a feature importance score for features involved in the construction of the given model. In essence, the importance score measures how valuable each feature was during the generation of the boosted tree model [9]. The more a feature is used to make decisions, the higher the relative importance is. The importance score is then determined for each feature based on the amount a feature split point improves the performance measure results, weighted by the number of samples the node is responsible for.

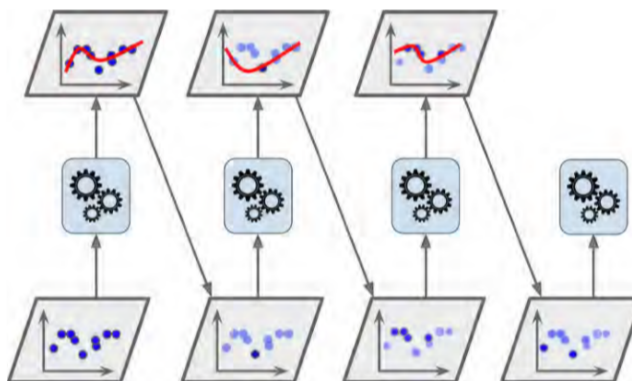


Figure 15: Gradient boosting. This figure is taken from [6]. Note that the points not lying on the plot are bold blue.

5.4. XGBoost and RNN Model Prediction Horizons

The prediction time steps from the present time step to a certain future time step is called the *horizon*. Figure 16 shows the difference between the RNN model and the XGBoost model horizons. Note that, the XGBoost model does not make a prediction for future time steps, but only for the present time step. Conversely, the RNN model has a horizon with future time steps. The number of past time steps used to make a model prediction is called the *lookback* and denoted by L .

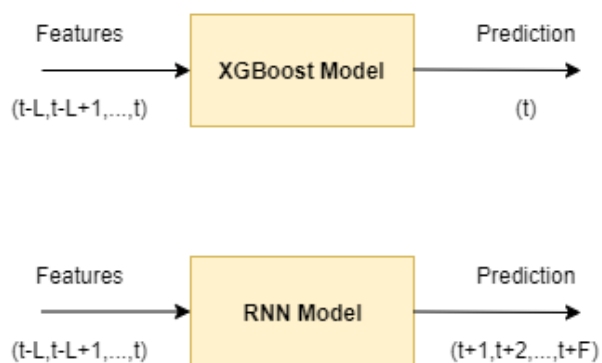


Figure 16: XGBoost and RNN model features and targets. L stands for the so-called lookback, which is the amount of past data steps that are used to make a prediction of the target. F denotes the amount of forecast steps.

Chapter 6: Methods

6.1. Introduction

In this chapter, the methods of implementing the two ML algorithms introduced in Chapter 5 are explained as preparation for the interpretation of results featured in the following chapter. Firstly, the inputs and output of the desired model are identified, based on the importance of the variables to the water softening treatment process. Next, the data collection method is described, where a data time interval is decided upon. Subsequently, the data is pre-processed using theory specified in Chapters 3 and 4. Finally, the methods used in the ML prediction phase are described, where the algorithms described in Chapter 5 are implemented.

6.2. Identification of Inputs and Outputs

Before requesting drinking water softening data from the water company, the potential input(s) and output(s) are identified.

The selection of inputs and outputs is limited, due to a lot of pertinent variables in the given Water Treatment Plant's (WTP's) softening process not being measured. For instance, the hardness is not measured in the softening process, since there are no installed sensors in the softening process. The hardness is instead constantly measured at the end of the WTP treatment process by a sensor.

In Machine Learning, the output is commonly referred to as the *target*. The target was identified as the pH in the softening process, since it has promising properties for control (as seen in Chapter 2). Based on availability of data and relevance to the pH (output), the inputs are chosen as: caustic soda flow, bottom pressure, pellet-bed pressure, bed height, draining and seeding. The relevance of the variables was assessed, by discussing the pertinent data with a process engineer from the given WTP. The inputs are called *features* in the ML domain. A description of the features and target are as follows:

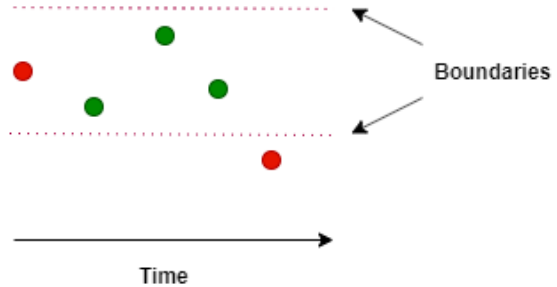
- **Flow** refers to the water flow rate travelling through the softening reactor [m^3/h].
- **Caustic Soda Flow (feature)** is the flow rate of the caustic soda dosing into the pellet-bed reactor [m^3/h].
- **Bottom Pressure (feature)** refers to the pressure measured at the bottom of the reactor [kPa].
- **Pellet-bed Pressure (feature)** refers to the pressure difference measured across the pellet-bed [kPa].
- **pH (target and RNN feature)** indicates the pH within the reactor.
- **Bed Height (feature)** represents the height of the pellet-bed [cm].
- **Draining (feature)** is the time at which the reactor drains an amount of pellets, i.e. when the pressure difference threshold across the reactor is exceeded.
- **Seeding (feature)** is the time at which the reactor is seeded.

The flow variable is removed from the dataset for ML after the analysis shown in Subsection 6.4.1 is carried out, which indicates that the caustic soda flow is strongly dependent on the flow.

6.3. Data Collection

6.3.1. Data Storage

The data is recorded live in the water company's system, using an interpolation method. It is possible to extract data from the system by specifying a time interval. This data is then subsequently transferred to an Excel document. Naturally, a smaller time interval for a requested period of time leads to more data points, thus costing more computation time to upload the data.



In order to reduce the amount of saved data on the water company's storage drives, interpolation is applied on the live data readings before being saved. The system adopted by the water company, records a value when it falls outside boundaries set by the WTP operator. For example, an operator may set the boundaries to 2% of the previously recorded value. Once a value falls outside the boundaries and is therefore recorded, interpolation is applied between the current recorded value and the previous one. This method of interpolation using boundaries can be viewed in Figure 17.

Figure 17: Data interpolation method. The red points denote the recorded data points and the green the disregarded data readings.

6.3.2. Time Interval Selection

Firstly, a day's worth of data with a time interval of one second is analysed, to ascertain which time interval would be appropriate to use for ML implementation. The time interval of one second seems to be unnecessarily low, because the data is interpolated (based on the method described in Subsection 6.3.1). Therefore, there is not a large loss of data information if a slightly larger time interval is selected. Deciding upon a one minute time interval, provides satisfactory retention of data information and low computation times.

Once the time interval, inputs and output are selected, a year's data is extracted from the system for ML implementation. A year's data provides enough information about the patterns of the softening process to train a ML model, since the seasons are not a large contributor to the dynamics of the water softening treatment process. For example, one of the most influential seasonal variables is the water temperature. The water temperature remains relatively constant throughout the whole of the WTP, due to there being always a large volume of water within the WTP at a given moment. Moreover, the WTP is insulated, so that the outside temperature does not have a substantial impact on the temperature inside the WTP. Typically, the temperature of the water in the WTP is largely determined by the temperature of the water at the source (collection point). On average the temperature is around 14°C throughout the year for the given WTP and is often two degrees higher or lower if the average is taken in the summer or winter respectively.

6.4. Data Pre-Processing and Data Analysis

The pre-processing and analysis in this section is conducted using the Python packages: NumPy, Pandas and StatsModels.

6.4.1. Erroneous Values and Variable Dependencies

Looking at Figure 22 on page 28, it is clear that the sensors are not recording sensible values (or are turned off) for irregular time intervals. For instance on 2 February, all the readings displayed in Figure 22 drop to zero, apart from pH, which instead falls in value considerably. The drop in the pH, could be explained by no caustic soda being dosed, which typically increases the pH. During these periods, it is also notable that no draining and seeding actions are recorded. This could be due to the reactor being switched off. For our machine learning algorithms, the corrupted data readings are removed from the dataset, since it does not describe the operational behaviour within the reactor. Removal of this uninteresting data, leads to retention of approximately two thirds of the data. The number of retained samples is considered sufficient to feed into the ML algorithms. The irregular time periods of corrupted readings span days. Thus, interpolating within these irregular time periods would not give an accurate representation of the actual values.

After removing the data with the corrupted sensor readings, we calculated the Pearson's correlation coefficient matrix for 2018 (seen in Table 7 in Appendix B) to understand the relationship of the variables with each other. It can be observed in Table 7 that there is strong correlation between the caustic soda dosing and the flow. This is due to the caustic soda flow being controlled by the flow in the given WTP.

Moreover, Figure 18 displays the caustic soda flow on the same plot as the flow over a few days in February. By observing Figure 18, it is evident that the caustic soda flow is strongly dependent on the flow. Hence, the water flow in the softening reactor is not included as a feature during ML training. Furthermore, caustic soda flow is retained, due to the variable having a greater impact on the pH in the softening reactor.

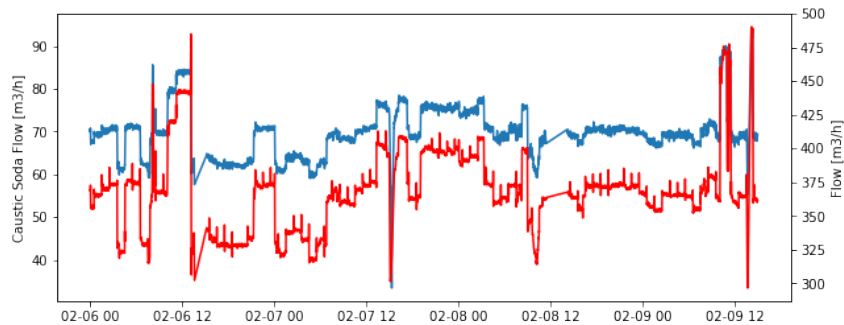


Figure 18: Caustic soda dosage flow rate and flow rate. The red plot is the flow and the blue is the caustic soda flow. The time axis increases in increments of 12 hours.

6.4.2. Feature Ranges

The minimums and maximums of the dataset after the erroneous are removed can be seen in Table 1. This suggests that the resulting ML model may only be able to give accurate outputs for inputs (features) within these boundaries, since the model is trained using a dataset that stays within these values.

Feature	Minimum	Maximum
Flow	200.6	502.9
Caustic Soda Flow	30	95.7
Bottom Pressure	5.1	25.5
Pellet-bed Pressure	11.3	35.6
pH	7.6	9.4
Bed Height	298.9	401.6

Table 1: Minimums and maximums of features to one decimal place.

6.4.3. Normalisation

Normalisation is applied to the *train* dataset, using the method explained in Section 3.3 and is only applied on the dataset used for the RNN ML algorithm, since the algorithm makes use of *activation functions*. Conversely, the XGBoost algorithm does not use activation functions and is based on decision trees. Therefore, normalisation of the dataset for the XGBoost algorithm is generally considered unnecessary [9].

6.4.4. Data Splitting

The data splits for ML algorithms are shown in Table 2. Note that, all the algorithm methods consisted of a 90% train and a 10% test data split, except for the train-validation-test RNN algorithm method. Based on the reasonably high number of samples available from the collected data, only 10% test data is required to evaluate the resulting models. Although, for the RNN Train-Validation-Test method, a 65% train, 30% validation and 5% test data split is picked, so that the hyperparameters can be altered during training to improve the model, based on the evaluation metrics of the validation data. This prevents overfitting on the test dataset, since the hyperparameters are adjusted on the basis of the validation dataset evaluation metric results, instead of on the basis of the test dataset evaluation metric results.

Data Splitting Method	XGBoost	RNN
Walk-Forward	90% train, 10% test	90% train, 10% test
Train-Validation-Test	90% train, 10% test	65% train, 30% validation, 5 % test

Table 2: Data splits for both algorithms: XGBoost and RNN.

6.4.5. Evaluation Metric Selection

The two evaluation metrics described in Section 4.6 (MSE and R-squared) are used in conjunction. Analysing the MSE evaluation metric gives insight into the accumulated error of the actual data points compared against the predicted data points. The R-squared provides information about the fit of the prediction relative to the actual data points.

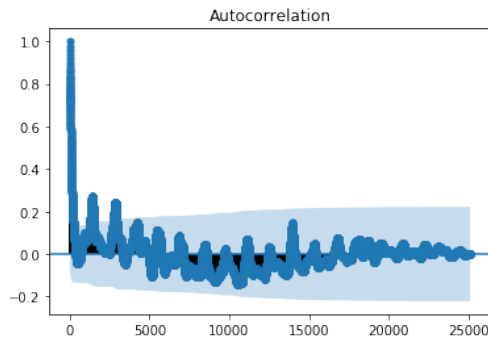
6.4.6. Output Variable Analysis

Before the prediction, the pH within the softening reactor is analysed. An analysis of the pH is necessary to understand the prediction outputs generated in the following prediction phase (discussed in the following section).

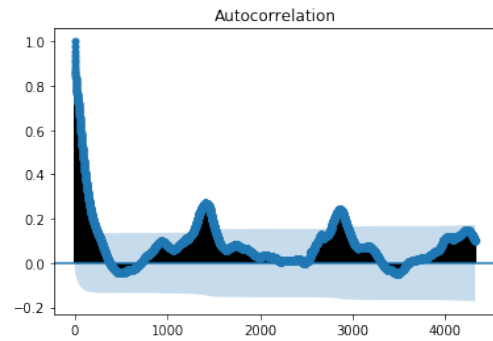
Draining of the pellets in the reactor is often shortly followed by seeding of the reactor (as seen in Figure 22 on page 28). Sometimes during these actions, the pressure difference across the pellet-bed increases drastically, since the pellet-bed has been disturbed. Furthermore, draining and seeding actions

appear to lead to an immediate decrease in pH, because the saturated pellets are drained and replenished with new sand. This increases the surface area, in which crystallisation of sodium carbonate takes place. Thereby, using up more caustic soda (with typically a pH of about 12).

The autocorrelation plot in Figure 19 of the pH in February 2018, shows that there is a relatively strong correlation with past time steps of data. Looking at the figure of three days worth of lags, it is evident that the autocorrelation has daily periodicity, where the x-axis consists of the number of minute lags. The daily periodicity can be explained by the draining actions in general occurring at approximately the same time of day for most days. This suggests that including past pH readings as a feature would improve the model produced through ML.



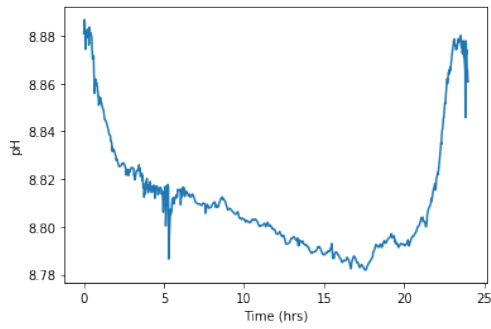
(a) Autocorrelation of the pH in February 2018.



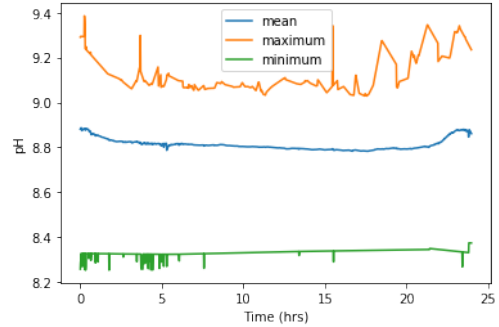
(b) Autocorrelation of the pH in February, displaying the three days worth of lags.

Figure 19: The light blue area represents the area bounded by the 95% confidence intervals, where the standard deviation is calculated using Bartlett's formula. The x-axis consists of the number of minute lags.

In Figure 20a the mean pH is taken over the period of a day, where time zero is the first time step after the draining action. This is calculated by compiling the instances of 24 hours of data after draining and taking the mean over all the samples for each time step in the 24 hours of compiled instances. In the first couple of hours after the draining action, the mean pH plummets, largely due to the sand seeding being added to the reactor. The plot then gradually continues to decrease until approximately 18 hours after the draining action. At about 22 hours, the pH increases significantly, since the pellets become saturated. Figure 20b depicts the mean pH bounded by the maximum and the minimum over the period of a day. The maximum and minimum are calculated for each time step, using once again the compiled 24 hours of data after a draining action. Furthermore, Figure 20b demonstrates the expected boundaries for a realistic pH prediction, i.e. to stay within the minimum and maximum boundaries over the course of the subsequent 24 hours after a draining action.



(a) Mean pH over one day. Time zero is the first time step after the draining action.



(b) Mean pH over one day bounded by the maximum and minimum pH. Time zero is the first time step after a draining action.

Figure 20

The box plot of the pH over the length of 24 hours is shown in Figure 21. Information about the key characteristics of a box plot can be found in Appendix B. It is clear that the pH peaks at about 9:00, since the majority of the draining actions occur at this time (see Figure 34 in Appendix B). Draining and seeding is commonly carried out during working hours, so that operators can oversee the actions and intervene if necessary. The IQR (Interquartile Range) is relatively large at 9:00, thus indicating that the pH readings are more skewed at this time. This could be the result of differing draining amounts. The draining action generally drains less than one percent of the pellet-bed in variable amounts. The *outliers* are the points below or above the bar at a corresponding hour in a box plot. There are many outliers in the following hours after 9:00, which is possibly due to draining actions sometimes occurring at other moments in the day or an increase in the flow rate, leading to a higher caustic soda dosage.

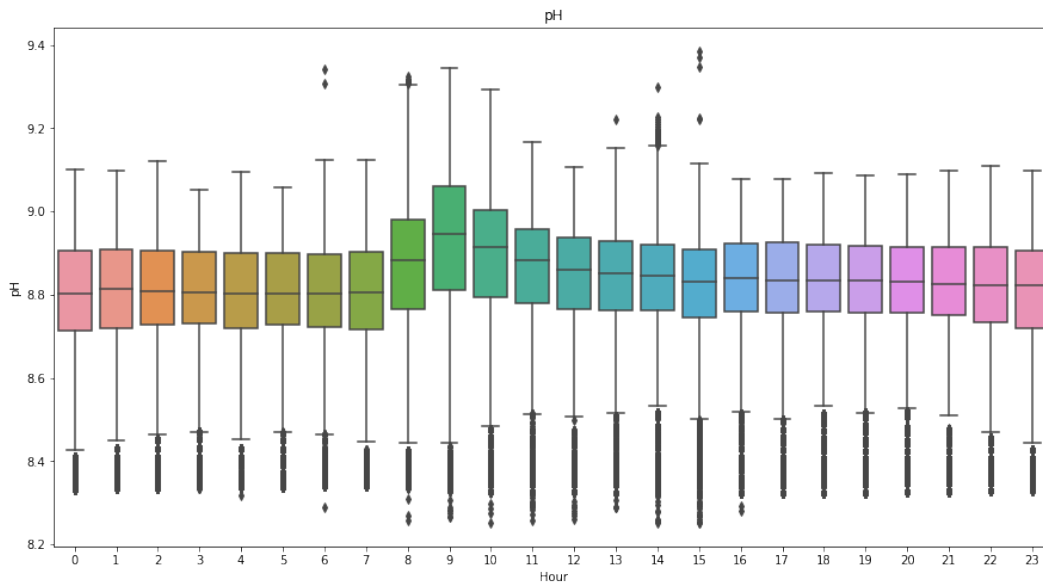


Figure 21: pH box plot over hours.

The maximum pH value is 9.4 and the minimum is 7.6, as observed in Table 1. Implying that an expected prediction should stay within these minimum and maximum boundaries.

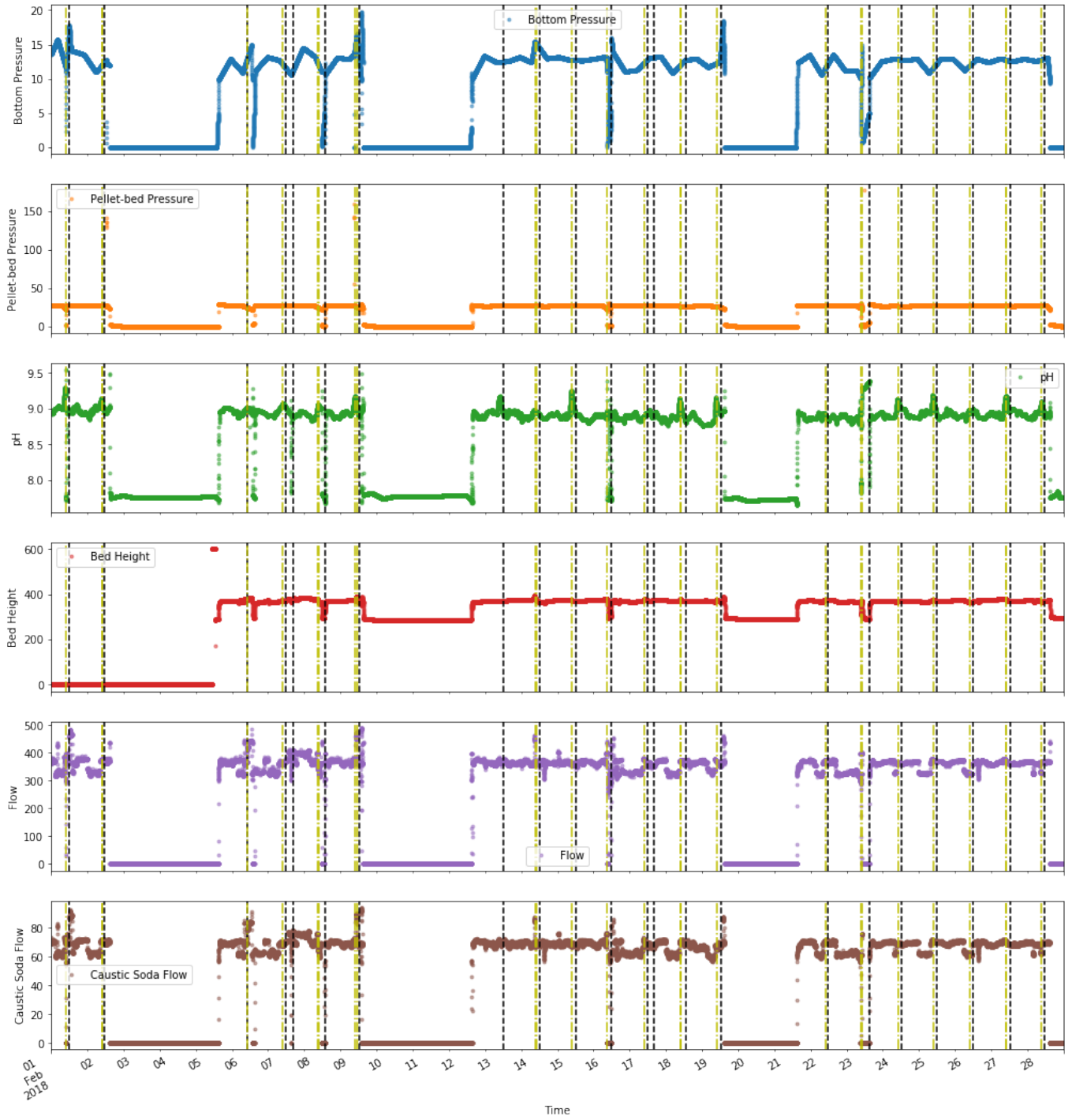


Figure 22: A month of data graphed. The vertical yellow spotted-dashed lines represent a draining action and the vertical black dashed lines represent a seeding action.

6.5. Prediction

6.5.1. ML Implementation

A. Sadad published his thesis about using a general ML framework for time series data [9]. For implementation of the RNN algorithm, the Python code (using Tensor Flow as *backend* and Keras as *frontend*) from A. Sadad has been adapted to give a greater *horizon span* (forecast length) and to use past data steps of the target (pH) as a feature.

Colaborative (subsidiary of Google) was used to execute the Python code, due to its free-of-charge GPU 12GB-RAM, which is significantly faster than the CPU RAM in a standard PC.

6.5.2. Hyperparameter Selection

The RNN structure adopted is as seen in Figure 10 on page 21, with seven features (inputs). The hyperparameter, *lookback*, refers to the amount of past time steps that are used in a ML algorithm (see Section 5.4). For the RNN implementation, a lookback of 180 minutes was set, based on a balance between computation time and evaluation metric performance. The other hyperparameter choices can be found in Appendix E for the walk-forward and train-validation-test model training.

Training is applied for the forecast lengths: $F=1[\text{min}]$, $4[\text{hr}]$ and $24[\text{hr}]$. This set of forecast lengths is selected to sufficiently test the limits of the RNN algorithm, i.e. the amount the evaluation metrics decline in performance as the forecast length is increased.

XGBoost is only able to make a prediction of the present target value based on present and past feature values (as seen in Figure 16). A lookback of 60 minutes (60 past data steps) was selected, based on a compromise between accuracy and computation times. The remaining value choices can be seen in Appendix E and were selected based on the Wastewater Treatment Plant (WWTP) XGBoost implementation by A. Sadad.

6.5.3. Supervised Learning

The predictions are made using the *supervised learning* training technique (described in Section 4.2) for both ML algorithms.

Chapter 7: Machine Learning Results

7.1. Introduction

In this chapter, the results of implemented Machine Learning (ML) algorithms, Recurrent Neural Networks (RNNs) and eXtreme Gradient Boost (XGBoost), are analysed using the evaluation metrics (defined in Chapter 3). The ML algorithms make use of the pre-processed data from Chapter 6. Based on the results, comparisons are drawn between the derived models from both of the data splitting methods: walk-forward and train-validation-test. The pH is selected as the target (output), therefore all the results in this chapter feature the pH as the output.

7.2. RNN Results

7.2.1. RNN Train-Validation-Test Model

Figure 24 depicts the prediction of the train-validation-test data split model, where the forecast horizon consists of one time step (i.e. $F=1[\text{min}]$), where F is defined in Section 5.4) and Figure 23 displays the last three hours of the prediction. Clearly, the prediction lags marginally behind the fluctuations of the actual data points. The prediction follows the pattern of the test data closely and is also able to follow sharp peaks. Training the model, gives a corresponding MSE equal to 0.0004 (4 d.p.) and an R-squared value of 0.9007 (4 d.p.). The relative MSE is equal to 0.0045% and is calculated by dividing the MSE value by the mean pH across the whole dataset used for ML and multiplying by 100.

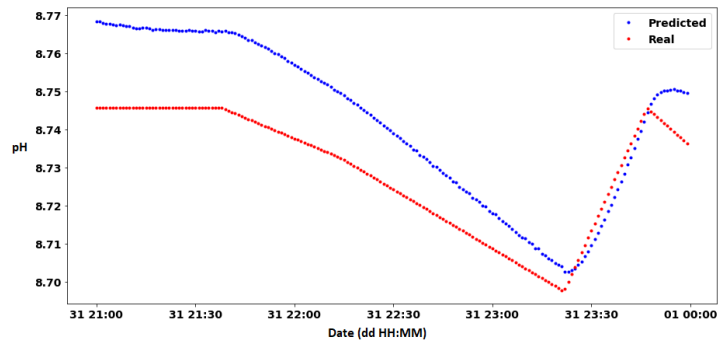


Figure 23: The last three hours of the RNN train-validation-test model prediction.

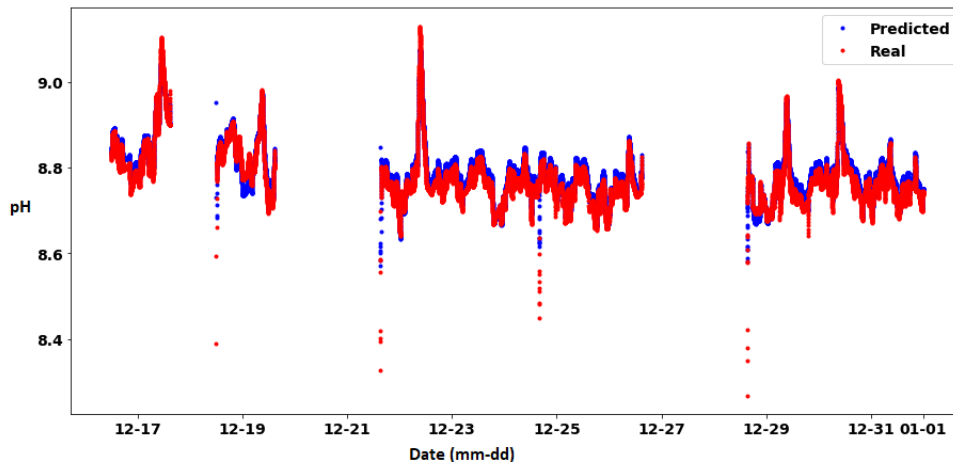


Figure 24: RNN train-validation-test model (with $F=1[\text{min}]$) prediction.

7.2.2. RNN Walk-Forward Models

Setting the prediction one time step into the future (i.e. $F=1[\text{min}]$, where F is defined in Section 5.4), gives the results in Figures 25 and 26. For more detail, the first three hours of models 0 and 4 are plotted in Figure 27. The prediction lags slightly behind the pattern of the actual data points. The gaps in data plots are a result of removing the corrupted data values. It can be deduced that the predictions follow the real data well, although model 4 appears to be slightly more shifted above the real data and consequently have a lower R-squared value (as seen in Table 3). In general, the predictions follow the pattern of the test data relatively closely, including the sharp peaks. For instance model 0 predicts a sharp peak on 25 October in the morning, which exceeds a pH of 9.

Model	MSE	Relative MSE	R-squared
0	0.0013	0.0148%	0.6373
1	0.0009	0.0102%	0.8801
2	0.0016	0.0182%	0.7269
3	0.0005	0.0057%	0.8638
4	0.0017	0.0193%	0.3000
5	0.0011	0.0125%	0.7678

Table 3: Evaluation metrics for the walk-forward RNN models to 4 decimal places.

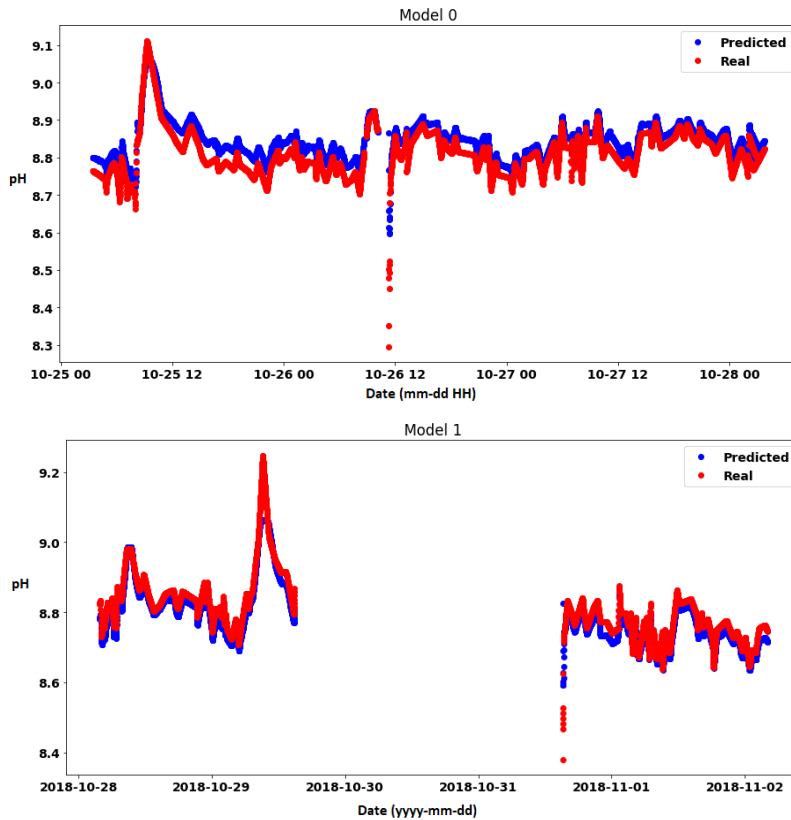


Figure 25: The first two RNN walk-forward models (with $F=1[\text{min}]$).

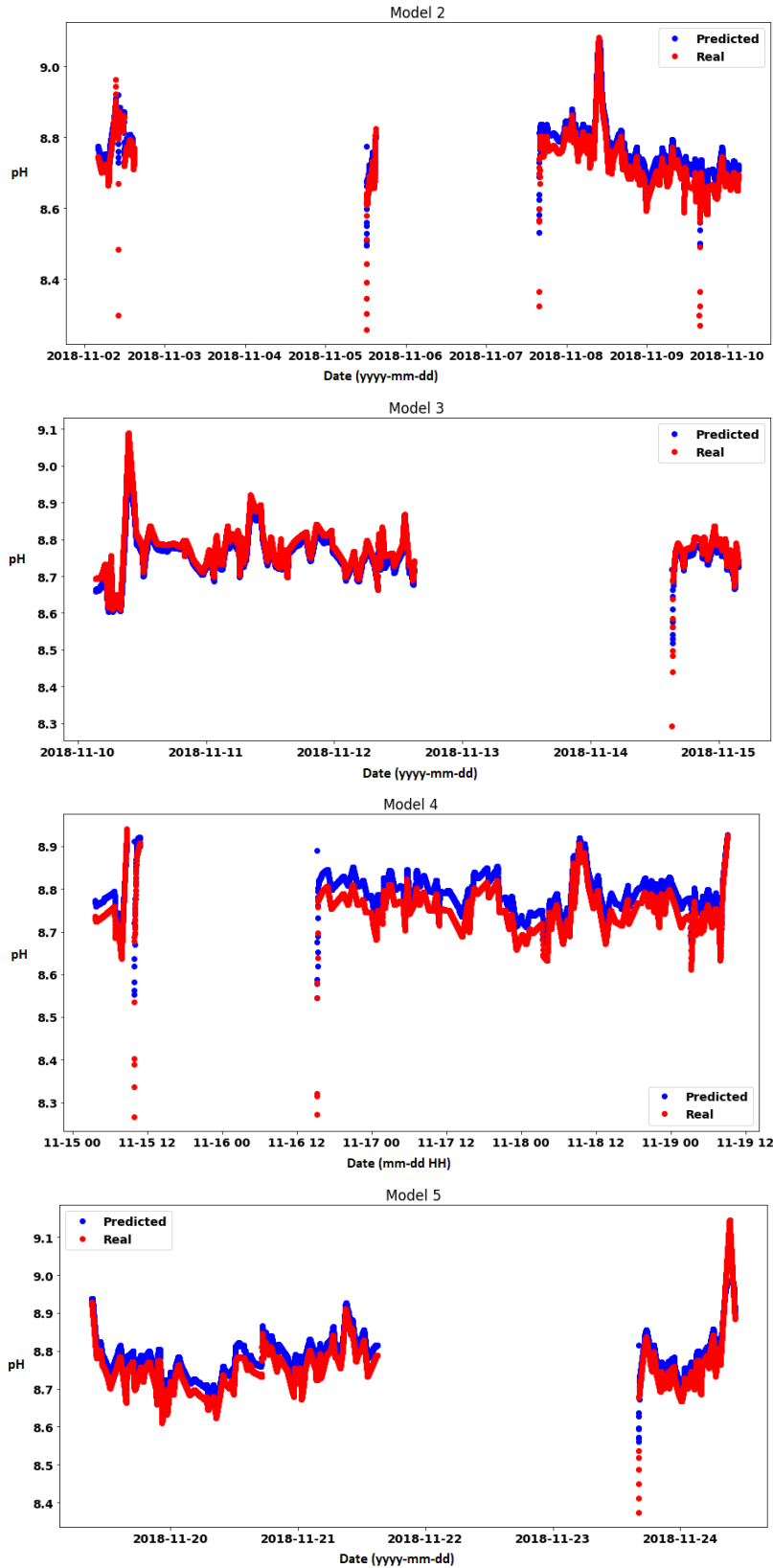


Figure 26: The last four RNN walk-forward models.

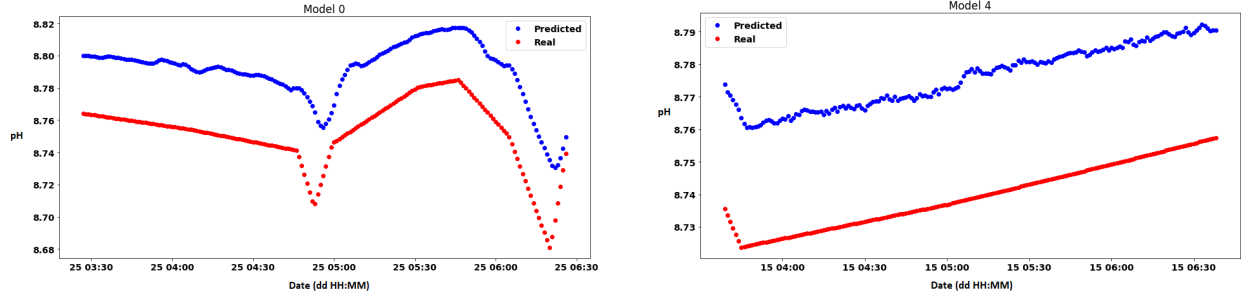


Figure 27: First three hours of the forecasts from the RNN walk-forward (with $F=1[\text{min}]$) models 0 and 4.

In Figure 28, the results of the walk-forward RNN model are shown, where the forecasting steps hyperparameter (F , as seen in Section 5.4) is set to 4 [hr]. In general, all eight models shown do not follow the pattern of the real data fluctuations closely. Furthermore, the predictions fluctuate more frequently than the interpolated test data. This is reflected in the R-squared scores displayed in Table 4. In particular, the predictions seem to be unable to follow the largest peaks in the pH, but instead remain relatively flat. For example model 1 does not predict the peak exceeding a pH of 9 at 10:00 on the 10th of November 2018. On the other hand, the errors between the prediction points and the real data points are generally relatively low. The small MSE values between the prediction and test data are accentuated by looking at the range of outputs, in the y-axes in Figure 28. The corresponding evaluation metric results can be viewed in Table 4.

Model	MSE	Relative MSE	R-squared
0	0.0026	0.0295%	-2.3583
1	0.0017	0.0193%	-801.8891
2	0.0032	0.0363%	-4.2461
3	0.0011	0.0125%	-0.5360
4	0.0023	0.0261%	-1.6060
5	0.0081	0.0919%	-1.5569
6	0.0008	0.0091%	-0.5401
7	0.0015	0.0170%	-0.7771
8	0.0022	0.0250%	-2.6563

Table 4: Evaluation metrics for the forecasting walk-forward RNN models to four decimal places. The forecast horizon is equal to 4 hours.

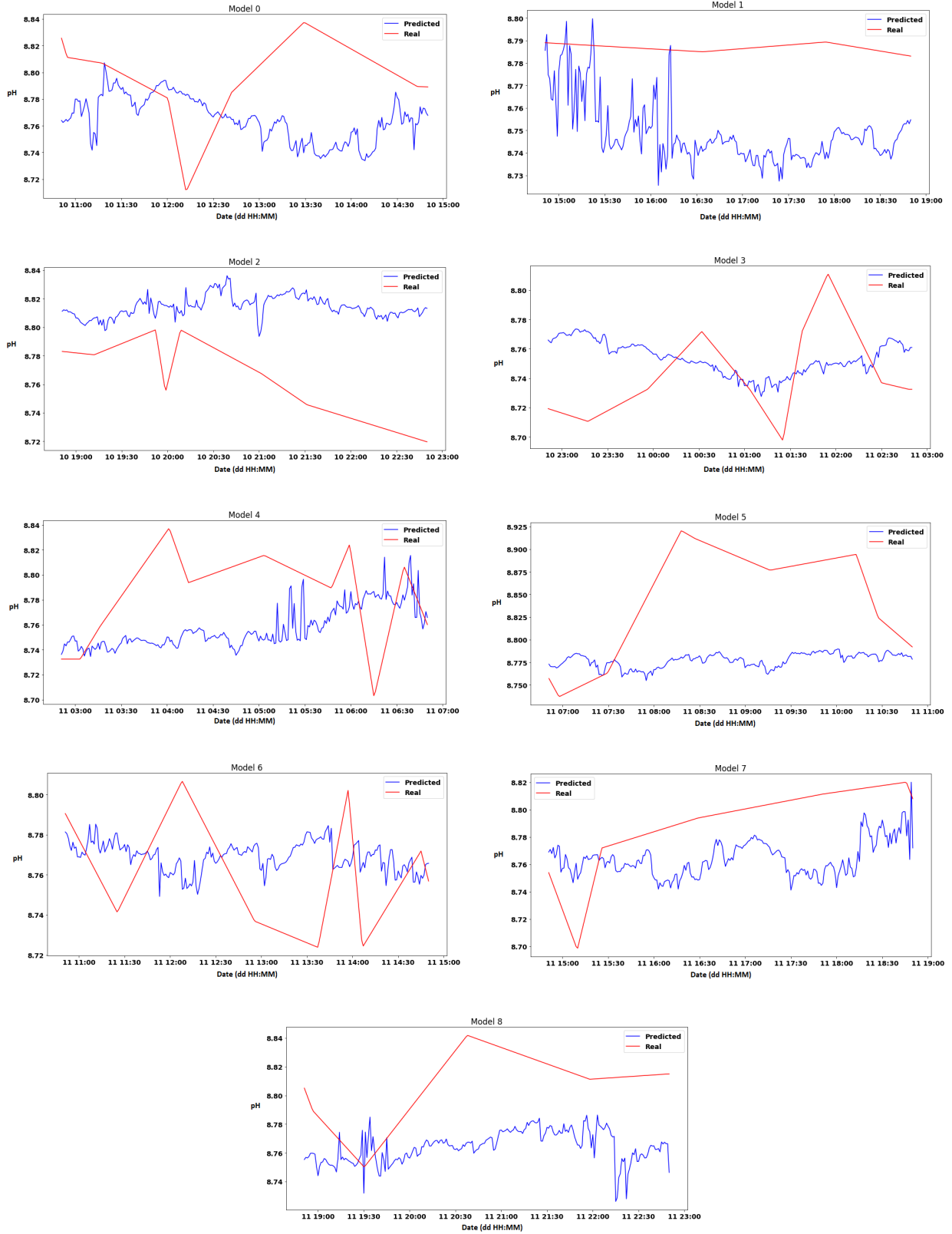


Figure 28: Walk-forward forecasting RNN model predictions, where $F=4$ [hr].

7.3. XGBoost Results

7.3.1. XGBoost Train-Validation-Test Model

The trained train-validation-test XGBoost model is displayed in Figure 29 and has a MSE value equal to 0.0057 (4 d.p.) and an R-squared value of -0.1871 (4 d.p.). In addition the relative MSE is equal to 0.0647%, which is calculated by dividing the MSE by the mean pH of the dataset (≈ 8.8097) and multiplying by a hundred. The predicted data as a whole does not follow the test data closely.

Figure 30 depicts the feature importance of the XGBoost train-validation-test model. The features for the present time step, pellet-bed pressure, draining and seeding, have the greatest impact on the resulting model. The most prominent past time steps from the most influential features, appear to be one, two, 58 and 59. For example, the 'bottom_press_59' represents the pressure at the bottom of the softening reactor 59 minutes ago and scores as the fourth most influential feature.

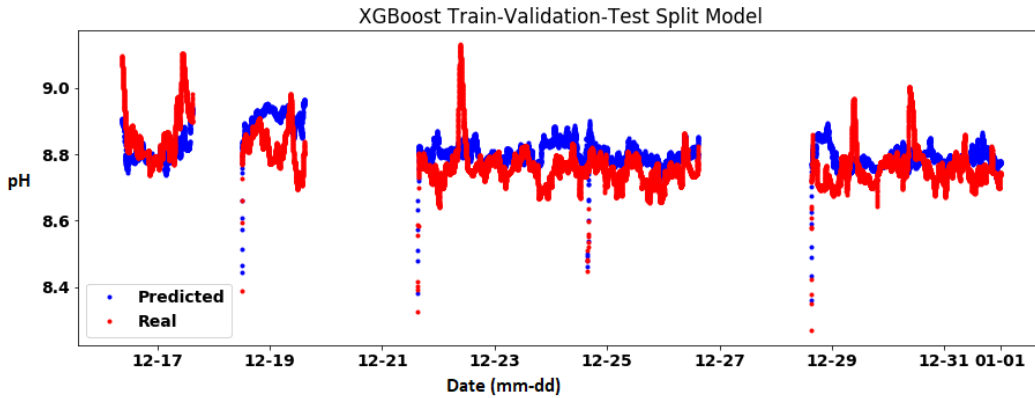


Figure 29: XGBoost train-validation-test split model prediction.

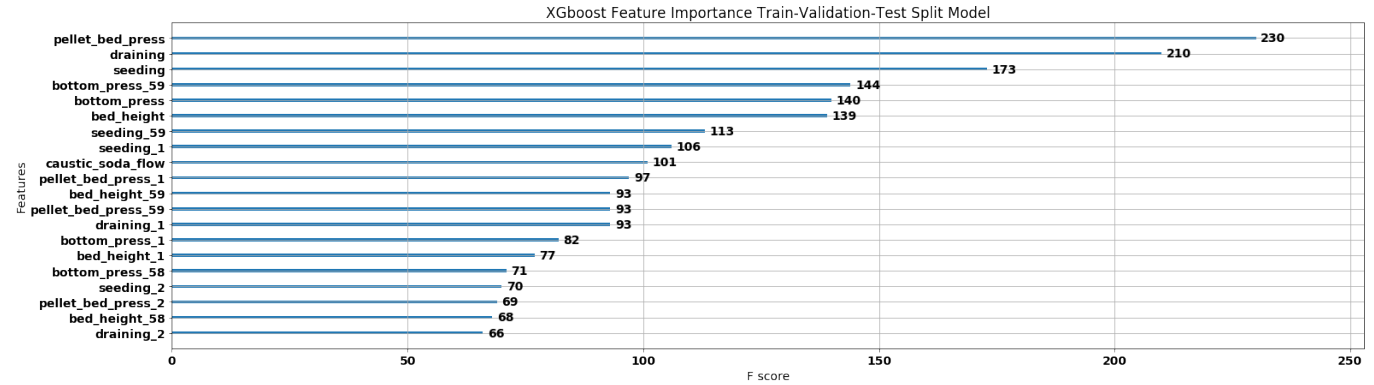


Figure 30: The feature importance of the train-validation-test XGBoost model. The 'pellet_bed_press' refers to the pellet-bed pressure difference feature for the present time step. An appended underscore followed by a number, denotes the past time step in minutes for that particular feature.

7.3.2. XGBoost Walk-Forward Models

The XGBoost walk-forward evaluation metric results are displayed in Table 5. All the R-squared values are negative, suggesting that the models do not fit the actual data closely. Alternatively, the MSE values are relatively low with respect to the pH scale, indicating a low error. This is confirmed by observing the relative MSE metric. The remaining walk-forward XGBoost results can be found in Appendix E.

Model	MSE	Relative MSE	R-squared
0	0.0176	0.1998%	-3.8138
1	0.0040	0.0454%	-0.2805
2	0.0049	0.0556%	-0.2389
3	0.0086	0.0976%	-0.5057
4	0.0276	0.3133%	-4.7629
5	0.0127	0.1442%	-7.4727
6	0.0043	0.0488%	-0.3530

Table 5: Evaluation metrics for the walk-forward XGBoost models to 4 decimal places.

Chapter 8: Discussion and Conclusions

8.1. Discussion

Comparing all the models displayed in Chapter 7 based on their evaluation metrics and output plots, it is evident that the Recurrent Neural Network (RNN) train-validation-test model (with $F=1$ [min]) performs the best, since its R-squared value is the largest (positive value) and its MSE value is the lowest (joint lowest with model 0 of the RNN walk-forward method). This could be the result of the model learning from a larger train dataset than the models produced using the walk-forward data split. Furthermore, looking at Figure 24, it is clear that the predictions closely follow the sharp peaks. This implies that the model can accurately model the behaviour of the pH when a draining action takes place. Conversely, a model with a forecasting horizon length of one minute is extremely limiting in practice, since a process engineer at a Water Treatment Plant (WTP) has only one minute to respond to changes based on the prediction. In addition, observing the walk-forward RNN and train-validation-test RNN results (with $F=1$ [min]) more closely (see Figure 23 and Figure 27), it can be noted that the prediction lags slightly behind the real data points. This may be caused by the model heavily relying on the present pH value to predict the next time step. Thus, the lag implies that the model is unable to predict the fluctuations at all, since it responds too late.

Clearly, the walk-forward forecasting RNN model (with $F=4$ [hr]) is unable to fit the real data points closely, by looking at Figure 28 and the R-squared results in Table 4. This could be due to the real data being interpolated, thereby giving it a flat profile, whereas the predictions exhibit frequent fluctuations. Conversely, the relative MSE remains under 0.1% for all of the walk-forward models produced (see Table 4). Therefore, indicating that the relative error between the predictions and the real data points is low. Increasing the forecasting horizon from a length of four hours to a length of 24 hours, gives in general slightly worse MSE results (see Table 8) and similarly negative R-squared values. These results are confirmed by looking at Figure 41 on page 62. Therefore, an increase of the forecast horizon from four to 24 hours does not seem to drastically worsen the results.

In Table 4, the extremely negative R-squared value for the RNN walk-forward model 1 can be justified by referring to Subsection 4.5.1. Extremely negative R-squared values are caused by significantly larger SS_{res} values than SS_{tot} values, i.e. the actual prediction performs significantly worse than the mean as a prediction. This is reinforced by inspecting the prediction displayed in Figure 28.

The poorer results of the XGBoost models, could be attributed to the model not incorporating past pH data as a feature or the model having a *lookback* of only 60 minutes (compared to a RNN lookback of 180 minutes). In particular, the R-squared values for the train-validation-test and walk-forward models are all negative (see Section 7.3), hence implying that the prediction values do not fit the real data points well. Moreover, although the relative MSE values for all the XGBoost models are low (less than 1%), they are in general not lower than the values derived from the RNN models.

8.2. Conclusions

In this section, the research questions posed in Section 1.4 are answered, making use of the results presented in Chapter 7.

8.2.1. Is it possible to develop a model of the softening treatment process of a WTP using Machine Learning (ML)?

The results featured in Chapter 7 suggest that an accurate model of the softening treatment process of a WTP can be constructed using the RNN ML algorithm (with $F=1$ [min]), based on the evaluation met-

rics. Moreover, all of the model predictions are considered feasible in the water softening reactor, since they stay within the maximum pH (9.4) and minimum pH (7.6) of the dataset used to train the model. On the other hand, the model is considerably limited, since it is only capable of predicting one minute into the future and is heavily influenced by the pH value at the present time step. Thus, the model does not display predictive qualities, since the fluctuations lag behind the actual data points. In addition, when the forecast horizon length is further increased (e.g. $F=10$ [min] or 10 time steps), the resulting model R-squared values are substantially worse, due to more fluctuations in the prediction than in the actual data. Although, the MSE values remain relatively low. This may be caused by the actual data being interpolated according to the method presented in Subsection 6.3.1.

8.2.2. Is the data provided for this research sufficient to develop a model using ML?

It can be concluded that the data provided is insufficient to generate a good performing RNN model. In spite of the RNN models (with $F=1$ [min]) producing excellent evaluation metric results, the models are heavily reliant on the present pH value. Thus, the model outputs a lagged version of the actual data. Increasing the horizon length beyond one time step generally leads to a worse R-squared value. For instance, the four hour walk-forward RNN forecasts have negative R-squared values (see Table 4) and generally do not fit the trends of the actual data (see Figure 28). This suggests that the interpolated data does not provide the amount of detail required to generate a model that fits the actual data closely for a large forecast horizon. On the other hand, the MSE values remain relatively low with respect to the pH scale.

In general, the XGBoost models have low MSE values, but negative R-squared values, thereby indicating that the predictions do not fit the test data well. This could be due to a lack of data to learn from, interpolation of the actual data or an absence of the incorporation of past pH values as a feature in the model.

The data does not show the behaviour for instances when no draining occurs during the day, therefore the model is conditioned to expect a daily draining action. For instance, regardless of the input, the model might output a peak in the pH, which would correspond to a draining action, even if no draining occurs on that particular day. Whereas, when there is no draining action, you would expect the pH to continue to increase, since the pellets in the reactor are saturated, causing a decrease in the *crystallisation rate* in the softening reactor.

The models are generated based on feature data contained within the maximum and minimum boundaries seen in Table 1. Therefore, the model may not be able to effectively predict for feature data outside these boundaries. This could be limiting, when extreme feature values (outside the boundaries) appear in the system, since the model might not be able to predict a realistic value for the pH in the softening reactor.

8.2.3. Can the seeding and draining control be improved by developing a control strategy using the produced ML model?

Based on RNN model predictive capabilities, it may be possible to predict when the pH is going to rapidly increase and therefore in advance carry out a draining action. The XGBoost model is incapable of forecasting, thus this control method can only be implemented using a RNN model. Furthermore, it is advisable to set the forecasting horizon to about a couple of hours or greater (e.g. 4 hours), so that the peak in pH can be on time anticipated and a draining action carried out in advance. Implementing this control method would have a stabilising effect on the pH softening treatment, i.e. the method should remove sharp peaks in the pH. In addition, the control implementation should lead to a more efficient

process, since the pellets are drained before the crystallisation rate declines significantly. Therefore, increasing the average of the crystallisation rate within the softening reactor.

The RNN walk-forward model predictions, with a prediction horizon equal to four hours (results seen in Figure 28), do not fit the test data particularly well. Moreover, the predictions do not appear to predict the sharp peaks in the pH value. Therefore, the model is regarded as unsatisfactory to use for the suggested control strategy implementation.

Chapter 9: Recommendations

In this chapter, potential further work is outlined, which could improve the performance of the Machine Learning (ML) model (based on the *evaluation metrics*). In addition, suggestions for implementing the draining and seeding control strategy (described in Subsection 8.2.3) are explored.

9.1. Machine Learning (ML) Recommendations

It might yield better results if you split the data into subsets that span from slightly after a draining action until slightly before the next draining action. These subsets will differ in size, because the draining actions do not occur at a set time every day. The ML algorithms should be adapted to learn from subsets of data of differing lengths. The implementation using subsets, could have the effect of removing the impact of the daily draining control action on the resulting ML model, since the dataset does not contain data around the time of draining.

Using a softening reactor experiment setup, allows you to collect data for when no draining action occurs for a certain period of time or irregular draining times. For instance, a draining action could have a probability of 75 % of occurring on a given day at a random time within WTP operating hours. A draining action occurring during operation hours, ensures that the draining action is supervised. If there is a defect during the draining action or afterwards, the operators are able to oversee and intervene if necessary. Experimentation could be carried out in a reserve softening reactor. Collecting data with irregular and infrequent draining, prevents the model from being conditioned to expect a draining action and allows the ML model to learn from data with more information, i.e. data that includes behaviour when a draining action does not take place on a certain day. Furthermore, using the experiment setup gives the opportunity to collect data for extreme *influent*s. For example, an extreme influent could exhibit extremely low or high pH values, where the influent is the water entering the softening reactor.

Due to computation times, it is not feasible to update the model every time step (one minute), since it takes longer than one minute to train the model. Although, it is possible to approximate computation times of training a ML XGBoost or a RNN (depending on the algorithm selected) model and therefore decide upon a regular model update that does not conflict with the computation time. The computation time can be reduced by using a GPU to run the ML implementation. Regular updating of the model allows the new model to incorporate more recent data trends, since there are often changes in the operations carried out at a WTP, that lead to different trends in the data. For instance, the water collection point (source) could be switched to another.

It is recommended to implement the best performing model from this research on another softening reactor using data with the same time interval. The model will work only if the features and the target are kept the same, with the same corresponding units as the dataset used to train the ML model. Another reactor from the same WTP, where the softening reactor has the same specifications and the influent comes from the same collection point (source), should perform accurately. If the model does indeed perform well, then it is possible to implement the model at other WTPs, thereby testing the generality of the model. Although, if this is unsuccessful, another ML model with the correct data can be trained for the respective softening reactor. Each WTP has a unique set of sensors installed, resulting in a unique set of variables featuring in the dataset for each WTP. Therefore, the features and target(s) should be modified accordingly.

A hyperparameter grid search could lead to a model of greater accuracy, where multiple possible hyperparameter permutations can be tested and compared using the ML evaluation metrics. Naturally,

the more permutations that are tested, the higher the computation time. This implementation was not applied for this research, since it was deemed too time consuming to program the grid search in Python given the deadline for this research.

Multiple output variables (targets) may be desired from the ML model. For example, a control method within the WTP could require multiple inputs (outputs from the ML model). It is possible to train a multivariate output model by adapting the Python implementation.

It may be possible to obtain data that is not interpolated, thus providing more information about the dynamics of the system. WTPs often have access to systems that take live readings from certain sensors. Training on data that is not interpolated should lead to a more accurate ML model, since it learns from more 'informative' data.

ML could possibly train a better performing softening treatment model, if the ML algorithm uses a larger train dataset. More data can be obtained by retrieving data further into the past from the same softening reactor or making the time interval of the extracted data smaller. For example, some WTPs have access to three years (backtracking from the present time) of stored data. Furthermore, a better performing model could result from data that contains less corrupted values, because less data would be removed (or interpolated) and the resulting dataset would have greater continuity with respect to the data time steps. Data with less corrupted values could be obtained from a different softening reactor.

A sensor measuring hardness could be installed in the softening reactor. Subsequently, these readings could be used as a feature for training of the ML model. This would probably yield a more accurate ML model, due to the algorithm having a larger dataset to learn from. Conversely, after the installation of the sensor, it takes time to accumulate data readings, thus initially the feature would not provide much information. Moreover, a sensor can be costly and the WTP would probably be reluctant to pay for a new sensor. Furthermore, there are often many softening reactors in a WTP, thus many sensors would need to be purchased to standardise all the softening reactors in the whole WTP.

9.2. Draining and Seeding Control Recommendations

A softening reactor experiment setup offers the chance to implement the control strategy proposed in Subsection 8.2.3. The running costs can be compared, for the following scenarios: the softening process with the implementation of the proposed control strategy and the softening process with the original control strategy. Note that this control method can only be acceptably implemented, if a forecasting model with a forecast horizon of at least two hours and satisfactory evaluation metric values can be generated.

It is hard to implement MPC (Model Predictive Control) to control the draining and seeding actions within a softening reactor, since a MPC cost function can not be easily formulated. This is due to the pellet softening reactor requiring a draining action each day within working hours, thus the amount of draining and seeding control actions can not be reduced without potential consequences. For instance, if a draining action is skipped on a certain day, then the reactor could become clogged leading to the process coming to a halt and costing most probably more than draining the pellets using the MPC method.

Bibliography

- [1] J. C. Van Dijk, J. Q. J. C. Verberk and P.J. De Moel (2018). *Drinking Water: Principles and Practices*. World Scientific Publishing Co Pte Ltd.
- [2] A. W. C. van der Helm, O. J. I. Kramer, J. F. M. Hooft and P. J. Moel (2015). *Plant wide chemical water stability modeling with PHREEQC for drinking water treatment*. International Water Conference: New Developments in IT & Water, Rotterdam, The Netherlands.
- [3] L. Rietveld (2015). *Softening*. Drinking water treatment, Delft University of Technology.
- [4] K.M. van Schagen (2009). *Model-Based Control of Drinking-water Treatment Plants*. Drinking water treatment, Delft University of Technology.
- [5] E. Chiou (2018). *An improved model of CaCO₃ crystallization*. Drinking water treatment, Delft University of Technology.
- [6] A. Géron (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.
- [7] C. M. Bishop (2006). *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA.
- [8] M. Nielsen (2013). *Neural Networks and Deep Learning*. Ebook available (06/09/2019) via: <https://github.com/antonvladyka/neuralnetworksanddeeplearning.com.pdf>.
- [9] A. Sadad (2019). *Reusable Machine Learning Framework for Predicting Future System Performance: A Comparison Study of Validation Strategy on WWTP Time Series Data*. Department of Electrical Engineering, Mathematics and Computer Science (EEMCS), University of Twente.
- [10] L. van den Hout (2013). *Modellering van de hardheidsreductie*. Hogeschool Utrecht, Institute for Life Sciences and Chemistry and Waternet, Sector Drinkwater, Afdeling Productie Team Waterkwaliteit en Procesondersteuning.
- [11] M. A. Jobse (2013). *Modellering van de fluidisatie bij korrelreactoren*. Thesis, Hogeschool Utrecht, Institute for Life Science and Chemistry.
- [12] J. Brownlee (2016). *Overfitting and Underfitting With Machine Learning Algorithms*. Article is available (12/09/2019) via the following link: <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>.

- [13] G. P. Pulipaka (2016). *An essential guide to classification and regression trees in R Language*. Article is available (12/09/2019) via the following link: https://medium.com/@gp_pulipaka/an-essential-guide-to-classification-and-regression-trees-in-r-language-4ced657d176b.
- [14] T. Chen and C. Guestrin (2016). *XGBoost: A Scalable Tree Boosting System*. University of Washington.
- [15] R. S. Brid (2018). *Introduction to Decision Trees*. GreyAtom School. Article is available (16/09/2019) via the following link: <https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb>
- [16] P. Krause, D. P. Boyle and F. Bäse (2005). *Comparison of different efficiency criteria for hydrological model assessment*. *Adv. Geosci.* 5, 89-97. University of Washington.
- [17] X. Yuan (2009). *Model Validation and New Water Control Strategies in Drinking Water Treatment Plant Wim Mensink*. Department of Water Management, Sanitary Engineering Section, Faculty of Civil Engineering and Geosciences, Delft University of Technology.
- [18] T. Smith (2019). *Autocorrelation*. Article is available (30/09/2019) via the following link: <https://www.investopedia.com/terms/a/autocorrelation.asp>. Investopedia.
- [19] M. A. Jobse (2013). *Modelling van de fluïdisatie bij korrelreactoren*. Thesis, Hogeschool Utrecht, Institute for Life Science and Chemistry.
- [20] E. Dizdar, A. Dizdar, C. Dizdar and A. Downey (2014). *PURE-H2O Implementation of ECVET for Qualification Design in Drinking Water Treatment Plants and Sanitation for Pure Drinkable Water*. Orkon International Engineering Training Consulting Co. Inc.
- [21] S. Hochreiter and J. Schmidhuber (1997). *Long Short-Term Memory*, *Neural Computation* vol. 9: 1735-1780 pg. Faculty of Computer Science, Technical University of Munich and IDSIA.
- [22] N. Srivastava, G Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*, *Journal of Machine Learning Reserach* vol. 15: 1929-1958. Department of Computer Science, University of Toronto.
- [23] *Dutch Drinking Water Statistics 2017: From source to tap*. Vewin, Association of Dutch water companies.
- [24] A. W. C. van der Helm and L. C. Rietveld (2002). *Modelling of drinking water treatment processes within the Stimela environment*, *Water Science & Technology Water Supply*. DHV Water BV and Delft University of Technology.
- [25] M. Galarnyk (2018). *Understanding Boxplots*. Article is available (18/10/2019) via the following link: <https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51>.
- [26] A. Sivalingam (2019). *What is machine learning ?*. Article is available (29/11/2019) via the following link: <https://medium.com/swlh/what-is-machine-learning-9b569ff7858a>.
- [27] S. Sharma (2017). *Activation Functions in Neural Networks*. Article is available (03/12/2019) via the following link: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>.

Appendix A: Drinking WTP Example and Softening Process Background Information

A.1. Example WTP

In Figure 31, a diagram of a drinking water treatment process is presented. The water is pre-treated at Loenderveen before reaching the Water Treatment Plant (WTP). The raw water is predominantly seepage water from the Buthune polder, sometimes blended with Amsterdam-Rhine Canal water. The raw water is coagulated with ferric chloride (FeCl_3) and flocs are removed in horizontal settling tanks. Resulting in the removal of phosphate, suspended solids, heavy metals and Natural Organic Matter (NOM) [4]. Afterwards, sedimentation, nitrification of ammonium and biodegradation take place in a lake-water reservoir of 130 hectares. The retention time of the reservoir is approximately 100 days. The left over ammonium, suspended solids and algae are removed using rapid sand filtration. The resulting water is then transported more than 10 kilometers to the Weesperkarspel WTP.

At the drinking WTP Weesperkarspel, the first process applied is ozonation for disinfection and oxidation of micro pollutants and NOM. Resulting in an increase in the biodegradability of organic matter. Subsequently, softening pellet reactors are used to reduce the hardness of the water and then biological activated carbon (BAC) filters are used to remove organic matter and organic micro pollutants. Finally, the water passes through a slow sand filter for extra nutrient and suspended solid removal. The water is then stored in a clear water reservoir ready for distribution.

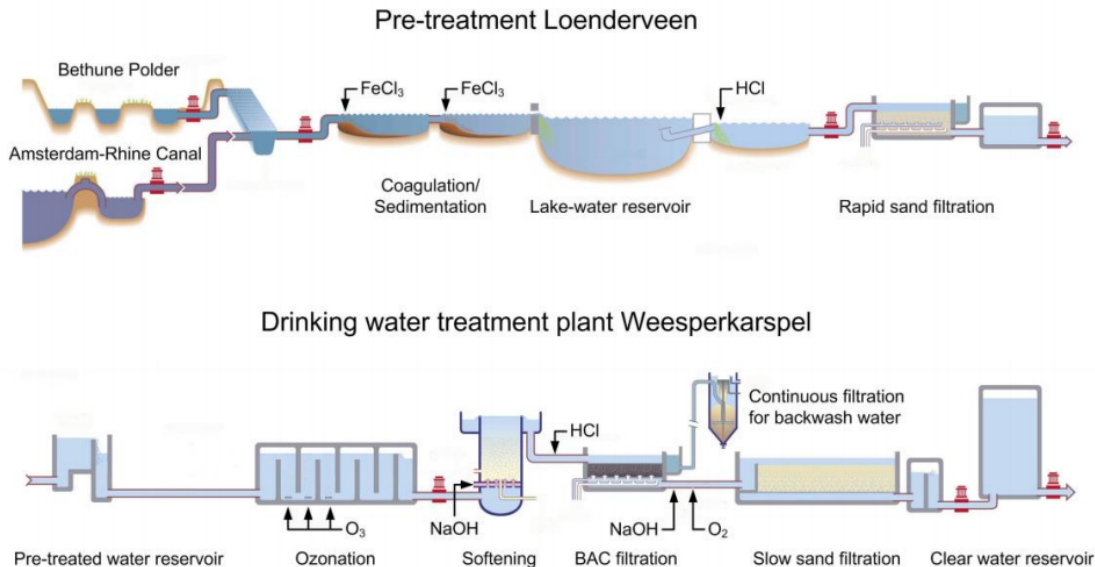


Figure 31: Diagram of the pre-treatment processes at Loenderveen and drinking water treatment at Weesperkarspel Water Treatment Plant (WTP) of Waternet [2]

A.2. Water Flux

The distribution of the particles in the reactor is dependent on the flow rate. A higher flow rate corresponds to a greater *water flux* in the reactor. We refer to the speed of the water travelling through the pellet-bed as water flux.

Figure 32 shows the impact of various water fluxes. The lowest speed on the left, has a compact layer of pellets accumulated on the circular plate in the reactor, due to the speed not being high enough to push the pellets into suspension. This scenario can cause clogging in the reactor. System scenario 2 in Figure 32 shows a situation where the water flux is large enough to displace the pellets, leading to the suspension of the pellets. Scenario 3 displays a greater distribution of pellets in the reactor, in reaction to an increase in the water flux. Finally, in scenario 4 the water flux is too high and leads to a large proportion of the pellets being flushed out of the reactor with the effluent.

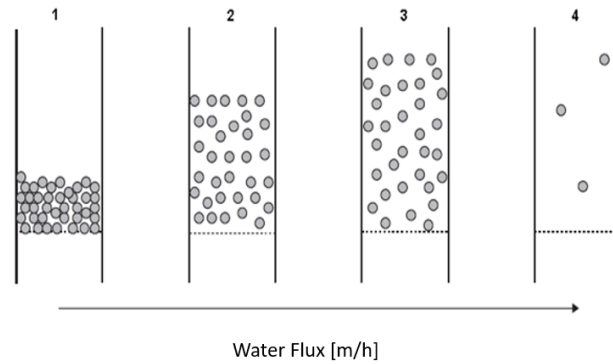
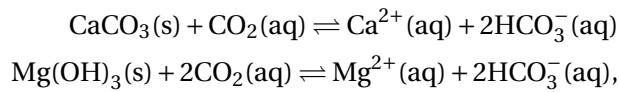


Figure 32: Water flux scenarios in the reactor. This figure is taken from [19].

A.3. Water Hardness Chemistry

The hardness of the water is the total amount of dissolved metal ions in the water. In practice, the hardness is determined by measuring the concentration of calcium and magnesium ions in the water, since they are generally the most abundant in the influent.

Hard water arises in places where, among other things, calcium and magnesium compounds dissolve under the influence of the carbon dioxide present in water [10]:



where (aq) indicates that the substance is dissolved in water and (s) signifies that the given substance is a solid. Conversely, when CO_2 is removed from the water, for example by heating, lime will precipitate. The hardness that can be removed by boiling is called *temporary hardness*.

The classification of hardness levels in the Netherlands can be observed in Table 6. Note that the total hardness is measured predominantly in millimoles per liter.

A.4. Bypass

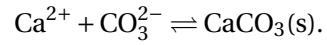
A pellet softening reactor is usually coupled with a *bypass* pipe. This bypass redirects a proportion of the raw water and reconnects downstream from the pellet softening reactor (and before the acid dosing), causing the softened water to mix with the raw water. The bypass contributes to preventing crystallisation in the following stages of the WTP, i.e. the bypass lowers the pH of the effluent. Furthermore, a bypass reduces the costs of softening, since the pellet softening reactor softens less water. The bypass mechanism can be viewed in Figure 1. Note that, for the softening reactor analysed in this paper, the bypass was set to zero percent (turned off).

Classification	[mmol/L]
Very soft water	<0.7
Soft water	0.7 - 1.4
Average water	1.4 - 2.1
Quite hard water	2.1 - 3.2
Hard water	>3.2

Table 6: Hardness classification in the Netherlands [11].

A.5. Calcium Carbonate Crystallisation Reaction

Calcium carbonate is one of the most broadly studied minerals and is investigated in studies involving geological, chemical and biological processes. The calcium carbonate crystallisation reaction is described by the following chemical equation:



This reaction often takes place on the softening pellets in a softening reactor of a WTP.

Appendix B: Data Analysis

B.1. Pearson's Correlation Coefficient Matrix

The Pearson's correlation coefficient matrix of the relevant variables of the drinking Water Treatment Plant (WTP) softening reactor used in this research is displayed in Table 7. The second highest correlation coefficient magnitude is between the bed height and the caustic soda flow, since caustic soda is used to facilitate crystallisation and crystallisation leads to thicker pellets in the reactor. Thick pellets weigh more and sink to a lower height in the reactor and consequently resulting in a higher bed height reading.

B.2. Box Plot

A boxplot is a visual representation of how the data is skewed. The graphic consists of (from left to right): "minimum", first quartile (Q1), median, third quartile (Q3) and the "maximum". The outliers are also shown in a boxplot. An example boxplot can be viewed in Figure 33.

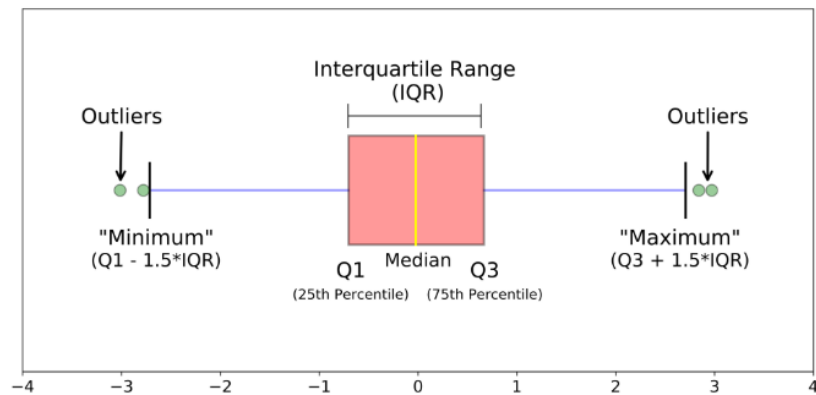


Figure 33: Boxplot. This image is taken from [25].

Variable	Flow	Caustic Soda Flow	Bottom Pressure	Pellet-bed Pressure	pH	Bed Height	Seeding	Draining
Flow	1	0.983	0.745	-0.119	-0.245	0.730	-0.104	-0.108
Caustic Soda Flow	0.983	1	0.724	-0.116	-0.220	0.737	-0.131	-0.134
Bottom Pressure	0.745	0.724	1	-0.094	-0.488	0.530	0.292	0.287
Pellet-bed Pressure	-0.119	-0.116	-0.094	1	0.003	-0.013	0.042	0.041
pH	-0.245	-0.220	-0.488	0.003	1	-0.066	-0.395	-0.392
Bed Height	0.730	0.737	0.530	-0.013	-0.066	1	-0.167	-0.168
Seeding	-0.104	-0.131	0.292	0.042	-0.395	-0.167	1	1.000
Draining	-0.108	-0.134	0.287	0.041	-0.392	-0.168	1.000	1

Table 7: Pearson's correlation coefficient matrix approximated to three decimal places.

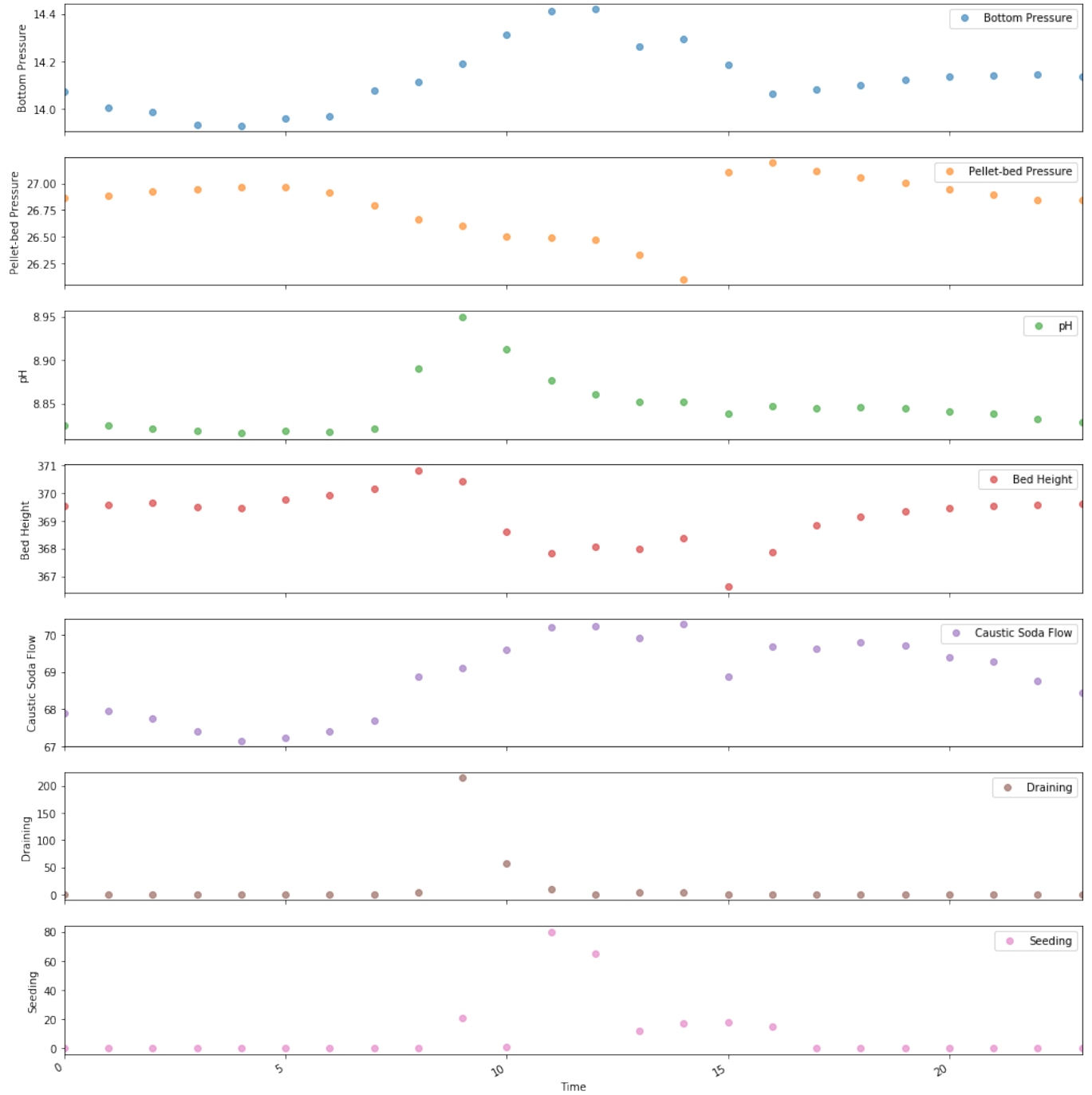


Figure 34: Hourly mean of data features and the sum of the data features, seeding and draining actions, per hour.

Appendix C: Machine Learning

C.1. Python vs Matlab®: Machine Learning and Control Theory Implementation

Before carrying out the analysis on the dataset used in this research, a particular software package was chosen to conduct the analysis. The choice was slimmed down to two candidates: Matlab® and Python. Eventually, Python was selected on the basis of the following comparisons between the two software packages:

- It is possible to obtain free extra RAM (memory) using Python in Colaborative. Using Matlab®, you are restricted to using the graphics card in your PC or your CPU, which can be considerably slower.
- There are more help forums for Python machine learning. For instance, Kaggle has many examples of machine learning algorithms being implemented using Python code. Matlab® on the other hand, has less information via online forums, due to it being less frequently used.
- Matlab® does not react as quickly to new machine learning developments as Python and thus the functionality of the machine learning toolboxes can be limiting. For instance, there is no Matlab® function for the XGBoost algorithm.
- Colaborative (provided by Google) is free to use, although you need to upload your dataset to Google Drive in order to use Colaborative. Some businesses may find this problematic, due to data confidentiality. Conversely, Matlab® requires you to pay for a subscription licence.
- Matlab® has a useful control theory toolbox called *Simulink*®. This allows you to effectively simulate and control theoretical systems using visual block diagrams. Although Python has a recently developed (in 2018) a new package SimuPy that is relatively similar to Simulink®, Simulink® is still regarded generally speaking to be the best amongst Systems and Control engineers. However, for this research Python is considered satisfactory to carry out the control methods, since the methods are not particularly complicated.

C.2. RMSProp Optimisation

RMSProp optimisation is often chosen over the standard gradient descent optimisation method, since it takes a more direct route to the optimum [6]. The standard Gradient Descent method, begins by going speedily down the steepest slope and then slowly down to the bottom of the valley. The RMSProp on the other hand, is able to prevent this, by scaling down the speed going down the steepest slope and taking a different route, that is more direct to the optimum point (bottom of the valley). RMSProp optimisation algorithm can be described by the following two equations:

$$s^{(i)} \leftarrow \beta s^{(i-1)} + (1 - \beta) \nabla_w C(\mathbf{w}^{(i-1)}) \otimes \nabla_w C(\mathbf{w}^{(i-1)}) \quad (3)$$

$$\mathbf{w}^{(i)} \leftarrow \mathbf{w}^{(i-1)} - \eta \nabla_w C(\mathbf{w}^{(i-1)}) \oslash \sqrt{s^{(i)} + \epsilon}, \quad (4)$$

where β represents the decay rate (often set to 0.9), \mathbf{w} the weight vector, \otimes symbolises the element-wise multiplication operation, \oslash the element-wise division operation, η is the learning rate, ϵ is a smoothing term to avoid division by zero (often set to 10^{-10}) and i denotes the time step. In addition, $\nabla_w C(\mathbf{w}^{(i)})$ is the partial derivative of C with respect to \mathbf{w} for the time step i .

C.3. Derivation of Backpropagation Equations

We can denote the weights of the Neural Network (NN) as w_{jk}^l using notation proposed by M. Nielsen [8], where the indices indicate that the weight connects the k -th neuron in the $(l-1)$ -th layer of the NN to the j -th neuron in the l -th layer [8]. Similar notation is employed for the *bias* b_j^l ; symbolising the j -th neuron in the l -th layer. Likewise, we express the activation as a_j^l and is described as follows:

$$a_j^l = h\left(\sum w_{jk}^l a_k^{l-1} + b_j^l\right), \quad (5)$$

where the sum is over all neurons k in the $(l-1)$ -th layer and h represents the activation function, which acts as a hyperparameter.

For simplicity, equation (5) can be rewritten in matrix form as follows:

$$a^l = h(w^l a^{l-1} + b^l).$$

Let $z^l \equiv w^l a^{l-1} + b^l$ and we refer to z^l as the *weighted input* to the neurons in layer l .

In order to update our weights when carrying out training, we need to define a cost function C . A popular choice is the *Mean Squared Error* (MSE) cost function:

$$C = \frac{1}{2N} \sum_x \|y(x) - a^L(x)\|^2,$$

where N is the total number of training samples, the sum is over the training samples x , $y(x)$ signifies the target output and L is the number of layers in the network and $a^L(x)$ represents the vector of activation outputs from the network and dependant on the input x . The aim is to minimise the cost function output (error) when evaluating the model using the test data.

We have assumed that $C = \frac{1}{n} \sum_x C_x$ with cost function C_x for an individual training sample. Therefore, for the MSE cost function $C_x = \frac{1}{2} \|y(x) - a^L(x)\|^2$. Let us suppose that the training sample is fixed and we drop the x subscript from the cost function. Thus, writing C_x as C .

Backpropagation in simple terms is about how changing the weights and biases in a network changes the cost function. Meaning mathematically that the partial derivatives, $\frac{\partial C}{\partial w_{jk}^l}$ and $\frac{\partial C}{\partial b_j^l}$, need to be computed. M. Nielsen [8] explains a four-step technique to calculate the backpropagation and is explained in the remainder of this Subsection.

Firstly, we define the error δ_j^l of neuron j in layer l by

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}, \quad (6)$$

where $z_j^l = \sum_{k=1} w_{jk}^l a_k^{l-1} + b_j^l$. Using the chain rule, equation (6) for the output layer is described as

$$\delta_j^L = \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L}, \quad (7)$$

where the sum is taken over all k neurons in the output layer. The output activation a_k^L of the k -th neuron is only dependent on the weighted input z_j^L for the j -th neuron when $k = j$. Thus, the terms in equation (7) disappear when $k \neq j$ and results in the following simplified expression

$$\delta_j^L \equiv \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L}. \quad (8)$$

Now, using equation (5) the second term in the product can be written as $h'(z_j^L)$. This gives

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} h'(z_j^L). \quad (9)$$

For our MSE cost function $C = \frac{1}{2} \sum_j (y_j - a_j^L)^2$, the partial derivative is $\frac{\partial C}{\partial a_j^L} = (a_j^L - y_j)$. Similarly as before, we denote the vector of errors associated with layer l as δ^l . Writing equation (9) in matrix form using the MSE cost function gives

$$\delta^L = (a^L - y) \cdot h'(z^L), \quad (10)$$

where \cdot symbolises the inner product.

Secondly, $\delta_j^l = \frac{\partial C}{\partial z_j^l}$ is rewritten in terms of $\delta_k^{l+1} = \frac{\partial C}{\partial z_k^{l+1}}$. This is achieved using the chain rule,

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}, \quad (11)$$

where the terms are interchanged. The first term in the resulting equation is further mathematically manipulated as follows

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} h(z_j^l) + b_k^{l+1}.$$

Taking the partial derivative, leads to

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} h'(z_j^l).$$

Substituting back into equation (11), yields the vector equation for the error δ^l in terms of the error in the next layer δ^{l+1} :

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \cdot h'(z^l). \quad (12)$$

Here, $(w^{l+1})^T$ is the transpose of the weight matrix w^{l+1} for the $(l+1)$ -th layer. Combining the two equations (10) and (12) it is possible to determine the error δ^l for any layer in the network, i.e. beginning with δ^L and working your way back calculating subsequently δ^{L-1} , δ^{L-2} , etc. Thus, *backpropagating* through the network.

Thirdly, $\frac{\partial C}{\partial b_j^l} \delta_j^l$ is derived using, once again, the chain rule:

$$\frac{\partial C}{\partial b_j^l} = \sum_k \frac{\partial z_k^l}{\partial b_j^l} \frac{\partial C}{\partial z_k^l} = \frac{\partial C}{\partial z_j^l} = \delta_j^l, \quad (13)$$

where $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$. Using the vector notation, equation (13) can be described as

$$\frac{\partial C}{\partial b^l} = \delta^l.$$

Lastly, the rate of change of the cost with respect to any weight in the network is calculated via the following formula:

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l. \quad (14)$$

The corresponding vector form for equation (14) is:

$$\frac{\partial C}{\partial w^l} = a^{l-1} \delta^l, \quad (15)$$

where the derivation is calculated in a similar way as the above equations, i.e. using the chain rule. An interesting side-effect of equation (15) is that when a^{l-1} is small ($a^{l-1} \approx 0$), the gradient term $\frac{\partial C}{\partial w^l}$ will also be small. This causes the weights to *learn slowly*, i.e. they do not change significantly during gradient descent implementation.

C.4. The Backpropagation Algorithm

The backpropagation equations derived in the previous section, make it possible to compute the gradient of the cost function, which is often used in gradient descent (described in Subsection 5.2.2) for the learning of a NN. The steps of the algorithm used to derive the gradient of the cost function, are as follows:

1. **Input x:** Compute activation a^1 for the input layer.
2. **Feedforward:** For each $l = 2, 3, \dots, L$ calculate $z^l = w^l a^{l-1} + b^l$ and $a^l = h(z^l)$.
3. **Output error δ^L :** Calculate the vector $\delta^L = \frac{\partial C}{\partial a^L} h'(z^L)$
4. **Backpropagation of the error:** For each $l = L - 1, L - 2, \dots, 2$ calculate $\delta^l = ((w^{l+1})^T \delta^{l+1}) \cdot h'(z^l)$.
5. **Output:** Finally, the gradient of the cost function is calculated using $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$ and $\frac{\partial C}{\partial b_j^l} = \delta_j^l$.

C.5. Logistic Activation Function

The logistic activation function is as follows:

$$\sigma(z) = \frac{1}{1 + e^{-z}},$$

where z symbolises the input.

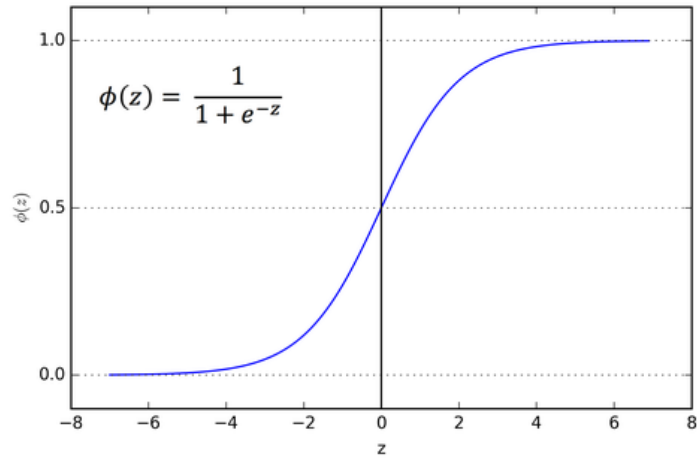


Figure 35: Logistic Activation Function. This image is taken from [27].

Appendix D: eXtreme Gradient Boost (XGBoost)

D.1. Regularisation Learning Objective

For a given dataset with n samples and m features (inputs) $\mathcal{D} = (x_i, y_i)$ ($|\mathcal{D}| = n, x_i \in \mathbb{R}^m, y_i \in \mathbb{R}$), a tree ensemble model uses K additive functions to predict the output. The prediction is described as follows:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F},$$

where $\mathcal{F} = \{f(x) = w_{q(x)}\}$ ($q : \mathbb{R}^m \rightarrow T, w \in \mathbb{R}^T$) is the space of regression trees, q denotes the structure of each tree which maps a sample to the corresponding leaf index (includes the tree decision rules), T is the number of leaves in the tree and f_k corresponds to an independent tree structure q and leaf weights w . Unlike decision trees (or classification trees), each regression tree contains a continuous score on each of the leaves and is represented as w_i for the i -th leaf. An example tree ensemble model is featured in Figure 36. In this example, the function ϕ for an example input x_1 is equal to:

$$\phi(x_1) = f_1(x_1) + f_2(x_1) = 0.3 + 0.9 = 1.2,$$

where f_1 corresponds to the first decision tree and f_2 to the second.

In order to learn the set of functions used in the model, we minimise the following *regularised* objective function:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k), \quad (16)$$

where $\Omega(w) = \gamma T + \frac{1}{2} \lambda \|w\|^2$ and l is a differential *convex* (i.e. $\forall \hat{y}_i, y_i \in \mathbf{X}, \forall t \in [0, 1] : l(t\hat{y}_i + (1-t)y_i) \leq tl(\hat{y}_i) + (1-t)l(y_i)$, where \mathbf{X} is a *convex set* in a real vector space and a convex set is a set of points, where every point on a line segment joining any two points within the set, lies within the set) loss function calculating the difference between the prediction \hat{y}_i and the *target* y_i . The second term Ω penalises the complexity of the model (the regression tree functions) and is considered a regularisation term that smooths the final learnt weights to avoid overfitting. A desired side-effect is that the regularised objective will tend to select a model employing simple functions and functions that can more accurately predict the target of new data. If the regularisation parameter λ is set to zero, the objective transforms back to the standard gradient tree boosting loss function.

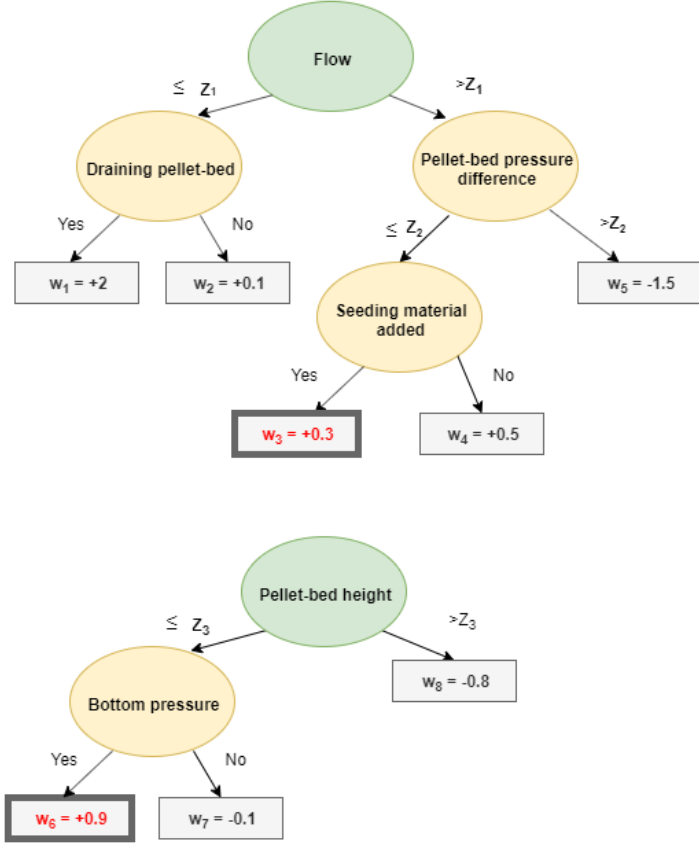


Figure 36: Example of a tree ensemble model, consisting of two decision trees. The red weights are the weights corresponding to a particular input x_1 .

D.2. Gradient Tree Boosting

The tree ensemble model described by equation (16) cannot be optimised by implementing the traditional methods in the Euclidean space, since the model employs functions as parameters. It is instead trained by applying addition. Let $\hat{y}_i^{(t)}$ be the prediction of the i -th instance at the t -th iteration. We add f_t and minimise the resulting objective function:

$$\mathcal{L}^{(t)} = \sum_{i=1}^n l(\hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i), y_i) + \Omega(f_t).$$

Therefore, the optimal f_t that most improves the model (seen in equation (16)) is added to the prediction in the loss function. The function f_t has T leaves. For a speedier optimisation the objective function can be written as follows using Taylor's series:

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^n \left[l(\hat{y}_i^{(t-1)}, y_i) + g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t),$$

where $g_i = \frac{\partial l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)}}$ and $h_i = \frac{\partial^2 l(y_i, \hat{y}_i^{(t-1)})}{\partial \hat{y}_i^{(t-1)2}}$ are respectively the first and second order partial derivatives of the loss function with respect to $\hat{y}_i^{(t-1)}$. We proceed by removing the loss function, resulting in:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^n \left[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t). \quad (17)$$

We define $I_j = \{i \mid q(\mathbf{x}_i) = j\}$ as the instance set of leaf j and rewrite equation (17), using the fact that $f(\mathbf{x}) = w_{q(\mathbf{x})}$ and by expanding Ω , as follows:

$$\begin{aligned} \tilde{\mathcal{L}}^{(t)} &= \sum_{i=1}^n \left[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 = \sum_{i=1}^n g_i w_{q(\mathbf{x}_i)} + \frac{1}{2} \sum_{i=1}^n h_i (w_{q(\mathbf{x}_i)})^2 + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2 + \gamma T \\ &= \sum_{i \in I_j} g_i \sum_{j=1}^T w_j + \frac{1}{2} \sum_{i \in I_j} h_i \sum_{j=1}^T w_j^2 + \frac{\lambda}{2} \sum_{j=1}^T w_j^2 + \gamma T = \sum_{j=1}^T \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T. \end{aligned}$$

For a fixed structure $q(\mathbf{x})$, the optimal weight w_j^* can be derived by taking the derivative with respect to the weight and rearranging, giving:

$$w_j^* = - \frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda},$$

The corresponding optimal objective value is then:

$$\tilde{\mathcal{L}}^{(t)}(q) = - \frac{1}{2} \sum_{j=1}^T \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T. \quad (18)$$

Subsequently, equation (18) can be used as a scoring function to measure the quality of a tree structure q . The score is similar to the impurity scores used to evaluate a decision tree (such as the impurity score calculated using the so-called *entropy function*), but instead applies for a wider range of objective functions. In most cases, it is computationally too expensive to find the optimal for every tree structure q . To avoid evaluating every possible q structure, the *greedy algorithm* is implemented. Briefly, the algorithm begins at a single leaf and iteratively adds branches to the tree [14].

Appendix E: XGBoost and RNN Implementation

E.1. XGBoost Results

Figures 37 and 38 display the resulting model predictions using the walk-forward data split method. Clearly, the predictions do not follow the pattern of the real data closely. This is confirmed in Table 5, where the R-squared evaluation metric values are negative. It is also evident, that the models are not able to predict the pH values in the region of a draining action, since the predictions generally do not spike significantly in value. On the other hand, the MSE evaluation metric values are relatively low.

The feature importance for model 4 is depicted in Figure 40. Clearly, the feature 'pellet_bed_press', denoting the pellet-bed pressure feature for the present time step, is by a long way the most important, followed by the present time step seeding, bed height, bottom pressure and draining features in importance. The most influential past data steps appear to be one, two, 58 and 59, where for instance a past time step of one indicates a measurement taken one minute ago.

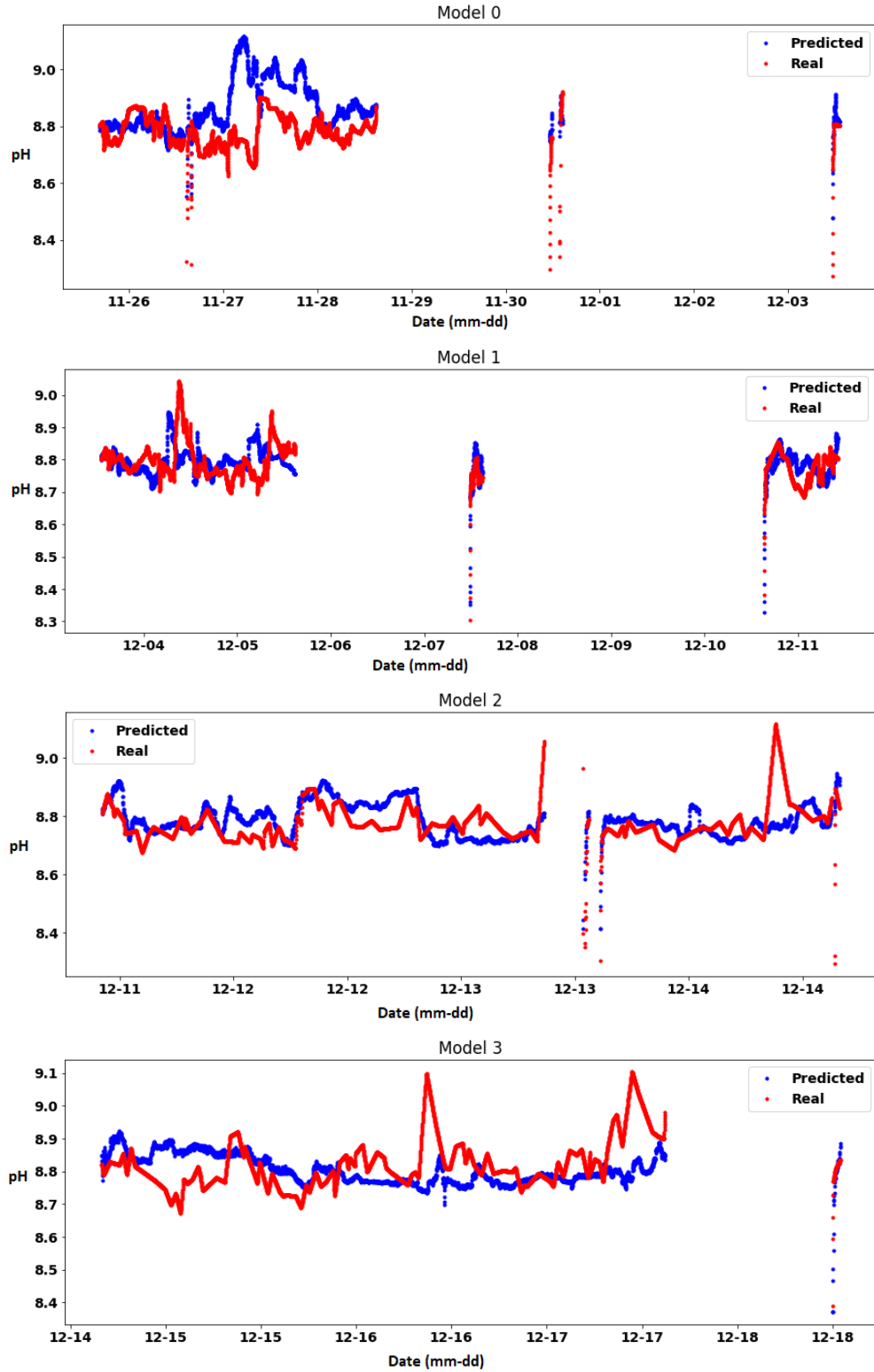


Figure 37: First four XGBoost walk-forward split model predictions on the same plot as the test data. The test data spans three days in total.

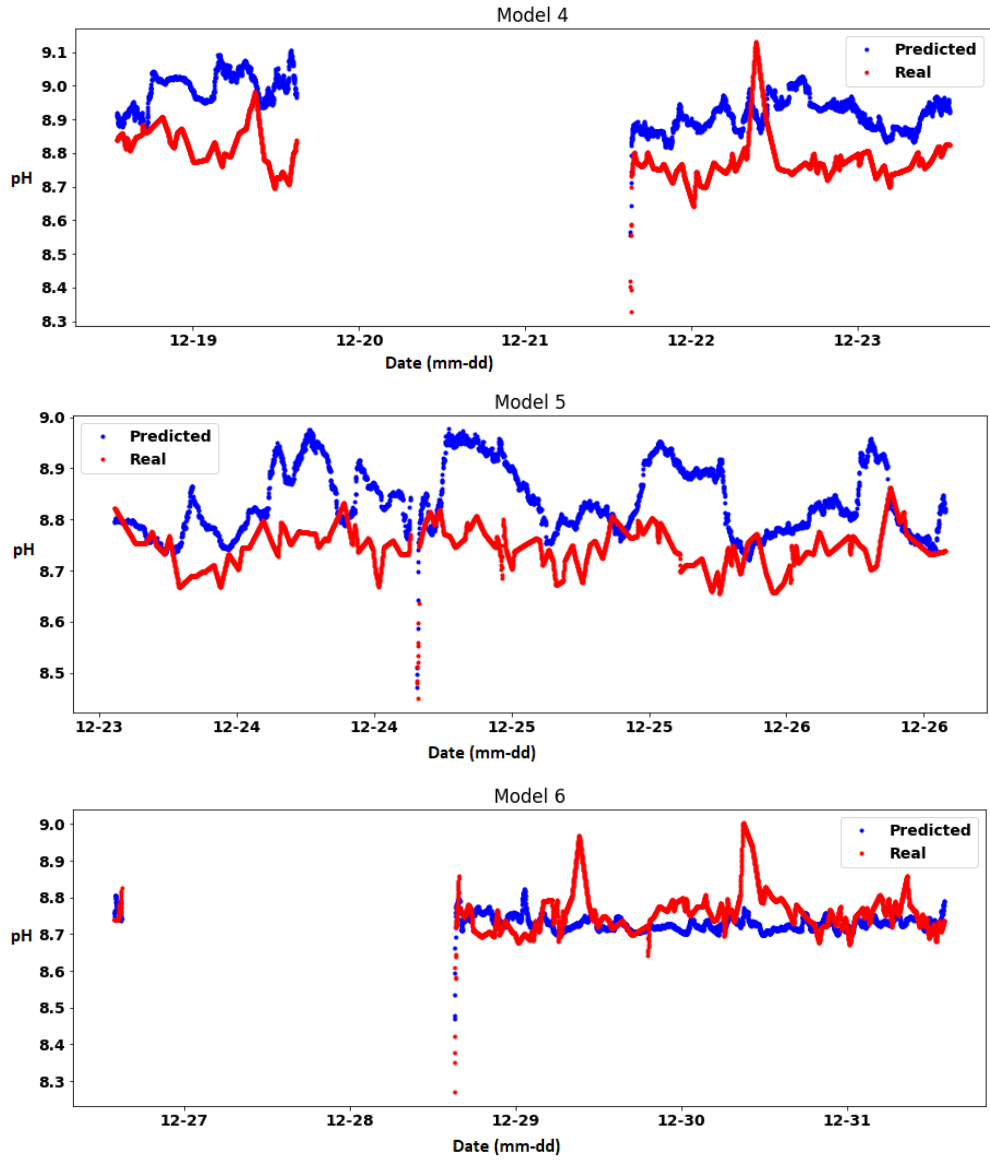


Figure 38: Last three XGBoost walk-forward model predictions on the same plot as the test data. The test data spans three days in total.

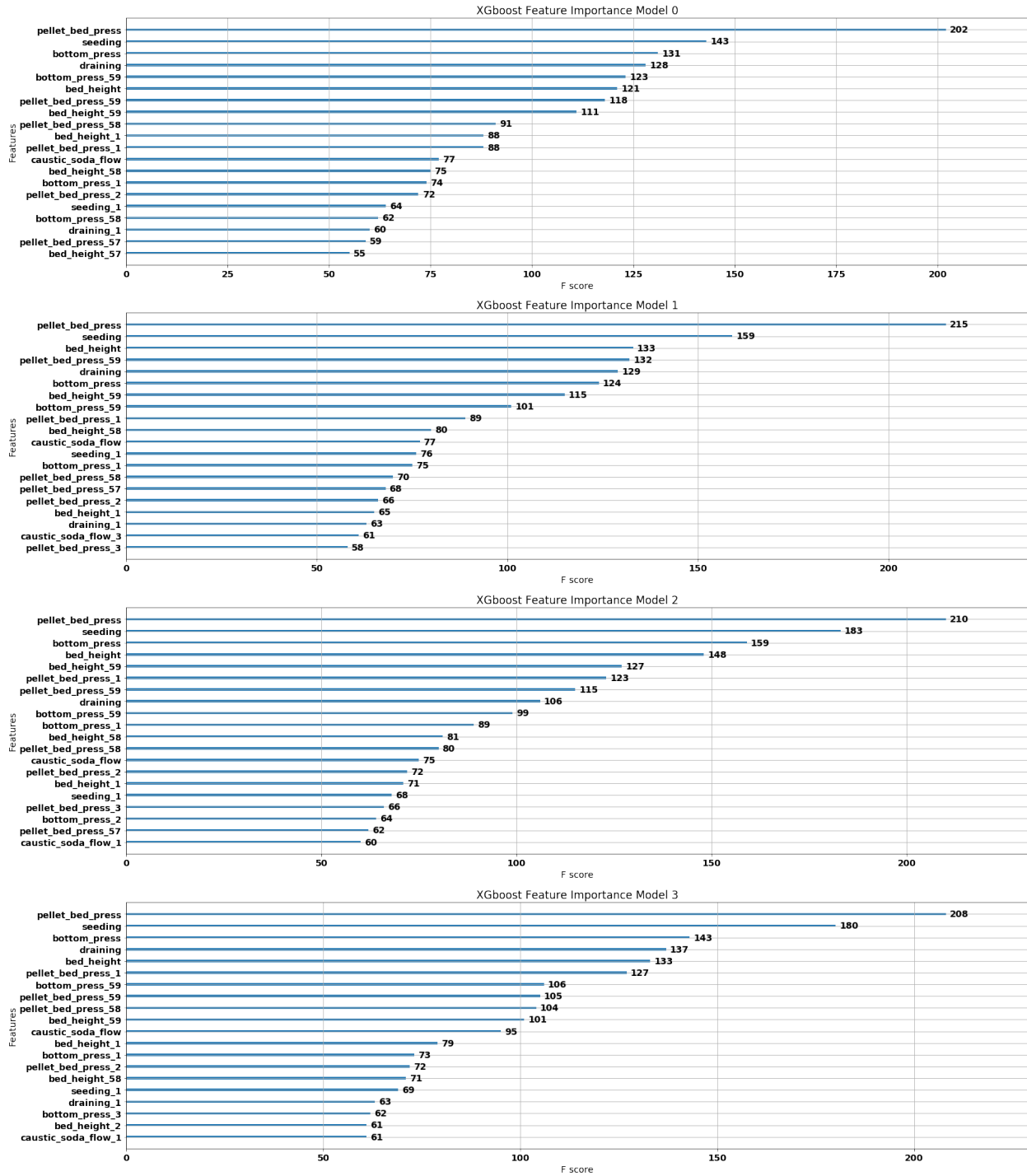


Figure 39: Feature importance of XGBoost models 0 to 3.

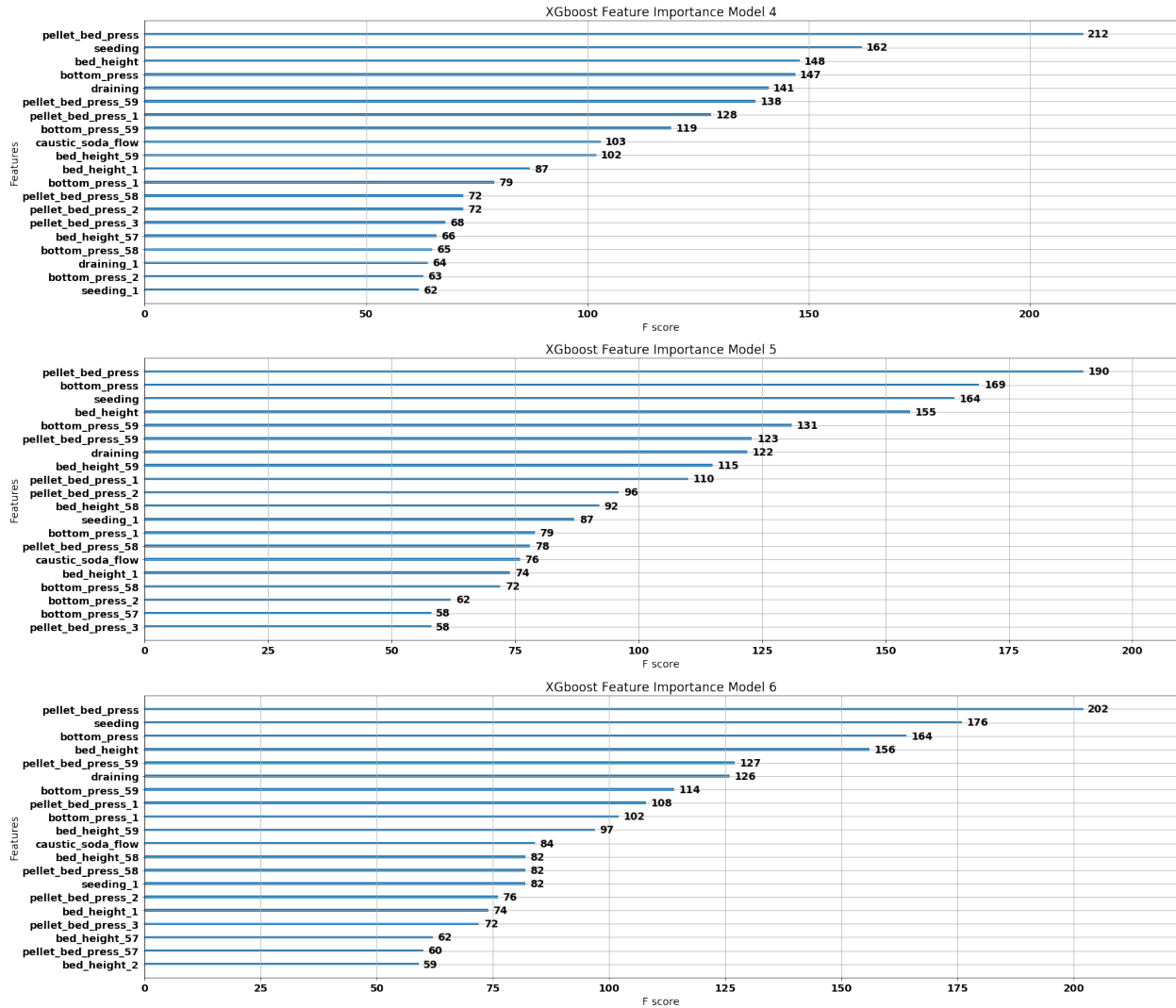


Figure 40: Feature importance of XGBoost models 4, 5 and 6.

E.2. RNN Results (F=24[hr])

Model	MSE	Relative MSE	R-squared
0	0.0120	0.1362%	-6.2375
1	0.0100	0.1135%	-0.1860
2	0.0035	0.0397%	-0.4968
3	0.0020	0.0227%	-0.1708
4	0.0030	0.0341%	-0.0843
5	0.0019	0.0216%	-0.1693
6	0.0031	0.0352%	-0.2907
7	0.0026	0.0295%	-0.1187
8	0.0038	0.0431%	-0.7773

Table 8: Evaluation metrics for the forecasting walk-forward RNN models to four decimal places. The forecast horizon is equal to 24 hours.

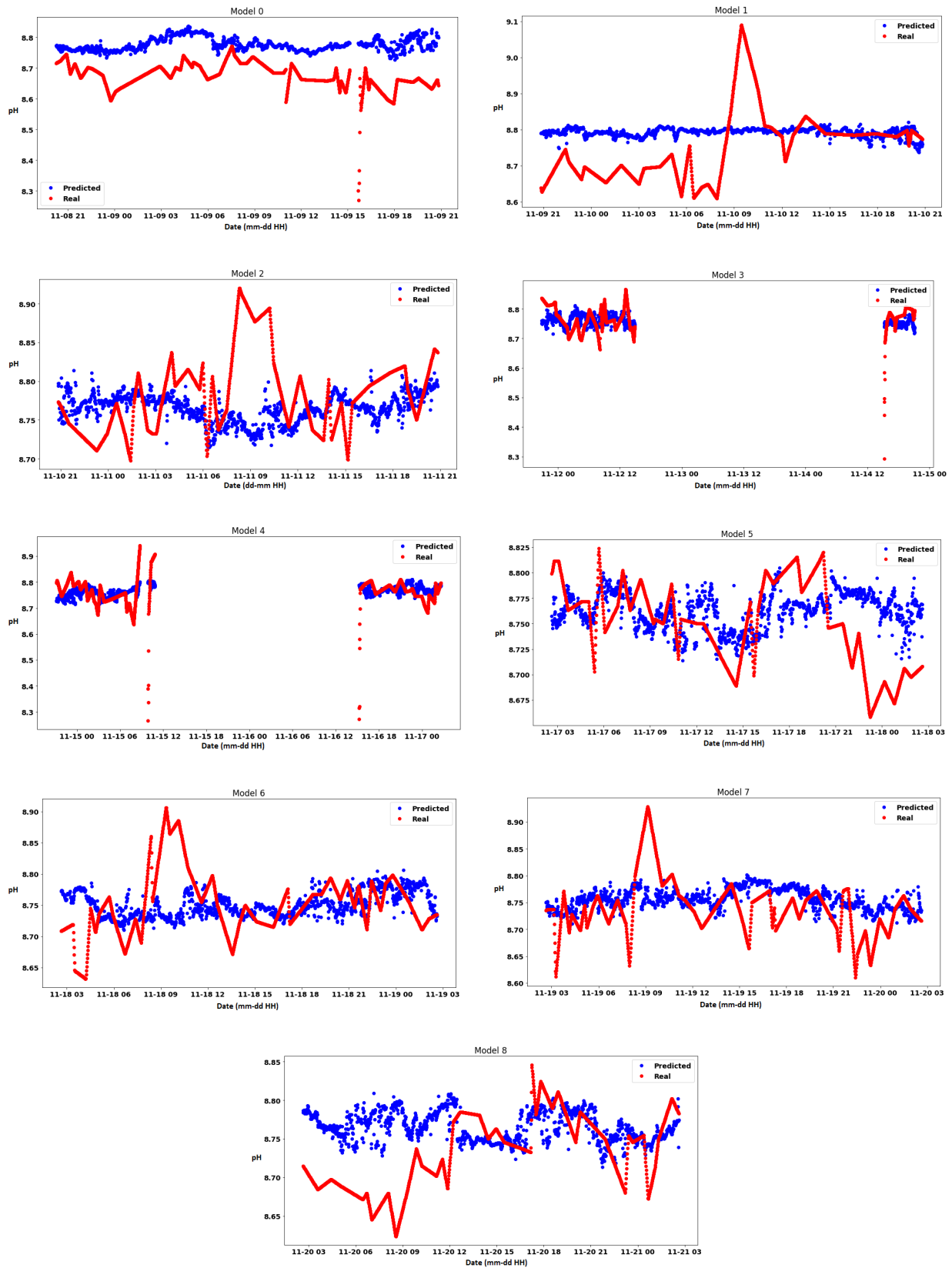


Figure 41: Walk-forward forecasting RNN model predictions, where $F=24$ [hr].

E.3. Hyperparameter Selection

E.3.1. XGBoost Hyperparameter Selection

For the XGBoost algorithm implementation the function *XGBRegressor* from the Python library *xgboost* was used. The following options are available within the function (according to the Python XGBoost documentation):

- *colsample_bytree*- Subsample ratio of the training instance, i.e. the fraction of columns to be random samples for each tree [9].
- *gamma*- Minimum loss reduction required to make a further partition on a leaf node of the tree.
- *learning_rate*- XGBoost learning rate.
- *max_depth*- Maximum tree depth. Used to control overfitting.
- *min_child_weight*- Minimum sum of instance weight needed in a child. Used to combat overfitting, since higher values hinder a model from learning relations.
- *n_estimators*- Number of trees to fit.
- *reg_alpha*- L1 regularisation term on weights.
- *reg_lambda*- L2 regularisation term on weights.
- *subsample*- Refers to the fraction of observations to be random samples for each tree.
- *seed*- The random number seed.
- *tree_method*- Specifies which tree method to use.

Options	Input
<i>colsample_bytree</i>	0.4
<i>gamma</i>	0
<i>learning_rate</i>	0.07
<i>max_depth</i>	3
<i>min_child_weight</i>	1.5
<i>n_estimators</i>	1000
<i>reg_alpha</i>	0.75
<i>reg_lambda</i>	0.45
<i>subsample</i>	0.6
<i>seed</i>	42
<i>tree_method</i>	'gpu_hist'

Table 9: XGBRegressor function hyperparameters used for implementation.

E.3.2. RNN Hyperparameter Selection

The RNN structure adopted is as seen in Figure 10, with seven features (inputs). The first LSTM from left to right is comprised of 64 neurons and the second 32 neurons. The dropout probability is set to 0.2. The amount of neurons and dropout probability are derived from A. Sadad's thesis for the Wastewater Treatment Plant (WWTP). Deviating these hyperparameters did not seem to give improved results. The regression output is the pH in the softening reactor and the Mean Squared Error (MSE) function was chosen for model training (see the backpropagation algorithm, in appendix C for more insight). Furthermore, when training the RMSProp optimiser (described in appendix C) was used.

E.4. Model Training Logs

E.4.1. RNN Walk-Forward Training (F=1 [min])

Three epochs and a batch size of 256 were used for training of the model, using the Batch Gradient Descent method (explained in chapter 5).

The Python log during training of the RNN walk-forward models (with F=1 [min]) is as follows:

```
Number of models for walkforward validation: 6
Training on index 189569:249569, Testing on index 249569:253889
Epoch 1/3
- 7s - loss: 2.5074 - mean_squared_error: 2.5074
Epoch 2/3
- 5s - loss: 0.0034 - mean_squared_error: 0.0034
Epoch 3/3
- 5s - loss: 0.0018 - mean_squared_error: 0.0018
MSE 0.0013159741401491626
R2 0.6372750930392224

Training on index 193889:253889, Testing on index 253889:258209
Epoch 1/3
- 7s - loss: 2.6694 - mean_squared_error: 2.6694
Epoch 2/3
- 5s - loss: 0.0031 - mean_squared_error: 0.0031
Epoch 3/3
- 5s - loss: 0.0016 - mean_squared_error: 0.0016
MSE 0.0009467924775900951
R2 0.8800728418928929

Training on index 198209:258209, Testing on index 258209:262529
Epoch 1/3
- 7s - loss: 2.1012 - mean_squared_error: 2.1012
Epoch 2/3
- 5s - loss: 0.0032 - mean_squared_error: 0.0032
Epoch 3/3
- 5s - loss: 0.0017 - mean_squared_error: 0.0017
MSE 0.0015642625093958199
R2 0.7268537541438583
```

Training on index 202529:262529, Testing on index 262529:266849

Epoch 1/3

- 7s - loss: 2.1511 - mean_squared_error: 2.1511

Epoch 2/3

- 5s - loss: 0.0036 - mean_squared_error: 0.0036

Epoch 3/3

- 5s - loss: 0.0022 - mean_squared_error: 0.0022

MSE 0.00049519045882465

R2 0.8638080469281029

Training on index 206849:266849, Testing on index 266849:271169

Epoch 1/3

- 7s - loss: 3.0528 - mean_squared_error: 3.0528

Epoch 2/3

- 5s - loss: 0.0038 - mean_squared_error: 0.0038

Epoch 3/3

- 5s - loss: 0.0022 - mean_squared_error: 0.0022

MSE 0.0017305357657509948

R2 0.29995530964421924

Training on index 211169:271169, Testing on index 271169:275489

Epoch 1/3

- 7s - loss: 2.6670 - mean_squared_error: 2.6670

Epoch 2/3

- 5s - loss: 0.0035 - mean_squared_error: 0.0035

Epoch 3/3

- 5s - loss: 0.0021 - mean_squared_error: 0.0021

MSE 0.0011098926523554079

R2 0.767833364861183

E.4.2. RNN Train-Validation-Test Training (F=1 [min])

The Python log during training of the RNN train-validation-test model (with F=1 [min]) is as follows:

Train on 200224 samples, validate on 92314 samples

Epoch 1/1

- 16s - loss: 1.4920 - mean_squared_error: 1.4920 - val_loss: 0.0024

- val_mean_squared_error: 0.0024

Train on 200224 samples, validate on 92314 samples

Epoch 1/1

- 14s - loss: 0.0032 - mean_squared_error: 0.0032 - val_loss: 0.0015

- val_mean_squared_error: 0.0015

Train on 200224 samples, validate on 92314 samples

Epoch 1/1

- 14s - loss: 0.0018 - mean_squared_error: 0.0018 - val_loss: 9.4053e-04

- val_mean_squared_error: 9.4053e-04

MSE 0.00044540128569551184
R2 0.9006795730436753

E.4.3. RNN Walk-Forward Training (F=24 [hr])

The Python log during training of the RNN walk-forward models (with F=24 [hr]) is as follows:

Number of models for walkforward validation: 9
Training on index 200699:260699, Testing on index 260699:262139
Epoch 1/3

- 8s - loss: 25.4802 - mean_squared_error: 25.4802

Epoch 2/3

- 6s - loss: 0.0743 - mean_squared_error: 0.0743

Epoch 3/3

- 6s - loss: 0.0084 - mean_squared_error: 0.0084

MSE 0.01203079266193375

R2 -6.237542567644389

Training on index 202139:262139, Testing on index 262139:263579

Epoch 1/3

- 8s - loss: 25.7666 - mean_squared_error: 25.7666

Epoch 2/3

- 6s - loss: 0.0786 - mean_squared_error: 0.0786

Epoch 3/3

- 6s - loss: 0.0084 - mean_squared_error: 0.0084

MSE 0.009982301908486433

R2 -0.18603822796927627

Training on index 203579:263579, Testing on index 263579:265019

Epoch 1/3

- 8s - loss: 25.6982 - mean_squared_error: 25.6982

Epoch 2/3

- 6s - loss: 0.0777 - mean_squared_error: 0.0777

Epoch 3/3

- 6s - loss: 0.0083 - mean_squared_error: 0.0083

MSE 0.003514358208861444

R2 -0.496832759297005

Training on index 205019:265019, Testing on index 265019:266459

Epoch 1/3

- 9s - loss: 25.5791 - mean_squared_error: 25.5791

Epoch 2/3

- 6s - loss: 0.0761 - mean_squared_error: 0.0761

Epoch 3/3

- 6s - loss: 0.0081 - mean_squared_error: 0.0081

MSE 0.0019810804050362347

R2 -0.1708376011249737

Training on index 206459:266459, Testing on index 266459:267899

Epoch 1/3
- 9s - loss: 25.6876 - mean_squared_error: 25.6876
Epoch 2/3
- 6s - loss: 0.0764 - mean_squared_error: 0.0764
Epoch 3/3
- 6s - loss: 0.0079 - mean_squared_error: 0.0079
MSE 0.003047756802827608
R2 -0.08428679237062675

Training on index 207899:267899, Testing on index 267899:269339

Epoch 1/3
- 9s - loss: 25.6174 - mean_squared_error: 25.6174
Epoch 2/3
- 6s - loss: 0.0761 - mean_squared_error: 0.0761
Epoch 3/3
- 6s - loss: 0.0078 - mean_squared_error: 0.0078
MSE 0.0018545439072492323
R2 -0.16927476280236786

Training on index 209339:269339, Testing on index 269339:270779

Epoch 1/3
- 9s - loss: 25.5759 - mean_squared_error: 25.5759
Epoch 2/3
- 6s - loss: 0.0752 - mean_squared_error: 0.0752
Epoch 3/3
- 6s - loss: 0.0078 - mean_squared_error: 0.0078
MSE 0.0031274762687713872
R2 -0.2907473882620255

Training on index 210779:270779, Testing on index 270779:272219

Epoch 1/3
- 9s - loss: 25.5906 - mean_squared_error: 25.5906
Epoch 2/3
- 6s - loss: 0.0761 - mean_squared_error: 0.0761
Epoch 3/3
- 6s - loss: 0.0077 - mean_squared_error: 0.0077
MSE 0.0026181155149427268
R2 -0.11874536027543758

Training on index 212219:272219, Testing on index 272219:273659

Epoch 1/3
- 9s - loss: 25.7260 - mean_squared_error: 25.7260
Epoch 2/3
- 6s - loss: 0.0786 - mean_squared_error: 0.0786
Epoch 3/3
- 6s - loss: 0.0076 - mean_squared_error: 0.0076
MSE 0.00377561454611042
R2 -0.7773242802171785.

E.4.4. RNN Walk-Forward Training (F=4 [hr])

The Python log during training of the RNN walk-forward models (with F=4 [hr]) for the first nine models is as follows:

Training on index 202979:262979, Testing on index 262979:263219

Epoch 1/3

- 6s - loss: 25.2531 - mean_squared_error: 25.2531

Epoch 2/3

- 5s - loss: 0.0879 - mean_squared_error: 0.0879

Epoch 3/3

- 5s - loss: 0.0085 - mean_squared_error: 0.0085

MSE 0.0025936636032497518

R2 -2.358257007307579

Training on index 203219:263219, Testing on index 263219:263459

Epoch 1/3

- 6s - loss: 25.5175 - mean_squared_error: 25.5175

Epoch 2/3

- 5s - loss: 0.0967 - mean_squared_error: 0.0967

Epoch 3/3

- 5s - loss: 0.0084 - mean_squared_error: 0.0084

MSE 0.0016546123607099616

R2 -801.8891442934232

Training on index 203459:263459, Testing on index 263459:263699

Epoch 1/3

- 6s - loss: 25.3872 - mean_squared_error: 25.3872

Epoch 2/3

- 5s - loss: 0.0890 - mean_squared_error: 0.0890

Epoch 3/3

- 5s - loss: 0.0084 - mean_squared_error: 0.0084

MSE 0.003222422027738503

R2 -4.24612932990162

Training on index 203699:263699, Testing on index 263699:263939

Epoch 1/3

- 6s - loss: 25.5995 - mean_squared_error: 25.5995

Epoch 2/3

- 5s - loss: 0.0926 - mean_squared_error: 0.0926

Epoch 3/3

- 5s - loss: 0.0084 - mean_squared_error: 0.0084

MSE 0.0011201554964107648

R2 -0.5359855698656777

Training on index 203939:263939, Testing on index 263939:264179

Epoch 1/3

- 6s - loss: 25.3692 - mean_squared_error: 25.3692

Epoch 2/3

```
- 5s - loss: 0.0880 - mean_squared_error: 0.0880
Epoch 3/3
- 5s - loss: 0.0084 - mean_squared_error: 0.0084
MSE 0.0022737514387699775
R2 -1.6060138628962237
```

```
Training on index 204179:264179, Testing on index 264179:264419
Epoch 1/3
- 6s - loss: 25.4679 - mean_squared_error: 25.4679
Epoch 2/3
- 5s - loss: 0.0916 - mean_squared_error: 0.0916
Epoch 3/3
- 5s - loss: 0.0083 - mean_squared_error: 0.0083
MSE 0.008136325113143054
R2 -1.5568766810367651
```

```
Training on index 204419:264419, Testing on index 264419:264659
Epoch 1/3
- 7s - loss: 25.6191 - mean_squared_error: 25.6191
Epoch 2/3
- 5s - loss: 0.0929 - mean_squared_error: 0.0929
Epoch 3/3
- 5s - loss: 0.0083 - mean_squared_error: 0.0083
MSE 0.0007989311898692601
R2 -0.5401342870579247
```

```
Training on index 204659:264659, Testing on index 264659:264899
Epoch 1/3
- 7s - loss: 25.2936 - mean_squared_error: 25.2936
Epoch 2/3
- 5s - loss: 0.0885 - mean_squared_error: 0.0885
Epoch 3/3
- 5s - loss: 0.0083 - mean_squared_error: 0.0083
MSE 0.001480038286005462
R2 -0.7771419745709061
```

```
Training on index 204899:264899, Testing on index 264899:265139
Epoch 1/3
- 7s - loss: 25.2056 - mean_squared_error: 25.2056
Epoch 2/3
- 5s - loss: 0.0867 - mean_squared_error: 0.0867
Epoch 3/3
- 5s - loss: 0.0083 - mean_squared_error: 0.0083
MSE 0.0022336121114487164
R2 -2.65633232488729
```

E.4.5. XGBoost Train-Validation-Test Training

The Python log during training of the XGBoost Train-Validation-Test model is as follows:

Training on index 0:292898, Testing on index 292898:308314
MSE 0.005656752010293507
R2 -0.18710613171327273.

E.4.6. XGBoost Walk-Forward Training

The Python log during training of the XGBoost walk-forward models is as follows:

Number of models for walkforward validation: 7
Training on index 217482:277482, Testing on index 277482:281802
MSE 0.017627705529895576
R2 -3.8138070276619054

Training on index 221802:281802, Testing on index 281802:286122
MSE 0.0040437185770498575
R2 -0.2805481716503393

Training on index 226122:286122, Testing on index 286122:290442
MSE 0.004901782404743926
R2 -0.23885435514421127

Training on index 230442:290442, Testing on index 290442:294762
MSE 0.008562428893266845
R2 -0.5057377146450821

Training on index 234762:294762, Testing on index 294762:299082
MSE 0.02763251097375469
R2 -4.7628726770061425

Training on index 239082:299082, Testing on index 299082:303402
MSE 0.012681836268440192
R2 -7.472719909357579

Training on index 243402:303402, Testing on index 303402:307722
MSE 0.004293264937667505
R2 -0.35304925162849266.