Utrecht University

Master's Thesis in Game and Media Technology

Real-time Outlier Detection in Time Series Data of Water Sensors

Faculty of Science
Department of Information and Computing Sciences

Author:
Luuk van de Wiel

Thesis Number:
ICA-4088212

First Supervisor:
Dr. A.J. Feelders

Second Supervisor:
Dr. Ing. habil G. Krempl

Daily Supervisor:
Dr. D. Van Es

May 4, 2020

Abstract

We compare multiple methods for real time outlier detection in time series data of water sensors. We present an outlier detection pipeline for this purpose. Multivariate models as well as univariate models are compared empirically by adding simulated outliers to the data to assess model performance. Quantile regression performed by the multi layer perceptron model using the tilted loss function is apt to model time series in a multivariate approach, provided we have access to reliable, correlated time series. Univariate models like auto-regressive models can be useful for detecting specific kinds of outliers such as extreme values. We show that the models are able to detect realistic, real life outliers.

Acknowledgements

I would like to thank my first supervisor, Ad Feelders, for his guidance through each stage of the project. Our discussions were invaluable to my progress and pointed me in the right direction. I am also grateful to my second supervisor, Georg Krempl. Moreover, Arjan Peters and Toon Basten from Waterschap Aa en Maas helped me greatly with their domain knowledge. Practical impact of the project was also maintained because of our conversations. I am also thankful to Daan van Es for supervising my project at Ynformed on a daily basis. Lastly, I would like to thank my friends, my family and especially my girlfriend Nikki van Bommel for their support, which motivated me and increased my perseverance.

Contents

Chapter 1

Introduction

Outlier detection is an important application of machine learning, which can improve data quality and thus improve data-driven analysis and decision making. Simply put, an outlier is a fundamentally deviating observation. Outlier detection can be applied more specifically to sensors. In this application, domain experts can be warned about possible misbehaving sensors or about an adjusted sensor environment. Then, action can be undertaken. For example, sensor technicians could be sent to investigate equipment errors or investigate the local environment of the sensor. This could be a great improvement compared to the times before outlier detection was used, since data faults were only detected after a prolonged period of time, which could render much data unusable. Also, a manual process to detect outliers is much more time-consuming and possibly less well-defined and consistent than an automated approach.

Data validation is an important area for water authorities in the Netherlands. These regional government bodies are responsible for among others sewage treatment, dyke management and the management of water levels in waterways. A validation pipeline along with implementation advice for water authorities has been proposed by Versteeg and de Graaff [48]. It is stated there that the main value that can be added by validation is that policy advice can be made more reliable. Data is often the basis of such kind of advice, which can be improved in quality by using more correct data. Other advantages like better operational management and enhanced assessment and reporting of current management practices can also be realised. Moreover, dimensioning of measures (for example, in determining the size of needed water storage) can be improved.

Our focus is on real-time outlier detection where multivariate time series of water sensor data are taken into account. The sensor data consists of time series, which means that data is obtained at successive times, with fixed intervals between measurements. Usually, *multivariate* indicates that multiple variables are measured, which might be very distinct in nature. Besides that, it is also possible to simultaneously use values of the same variable of multiple sensors to determine whether outliers occur in a certain sensor. Different sensors can output time series that are closely correlated with each other. In such cases, we can use time series from one or multiple sensors to predict other sensor values. If a big difference between the predicted and observed value occurs, the value may be classified as an outlier[1]. It is important that outliers are detected in real-time, because it enables taking immediate action to resolve possible issues. This might be crucial in crisis situations. The real-time aspect can be achieved by using scalable models, which might be necessary as the existing data stream is continuous. This possibility to take immediate action is an improvement over only being able to historically clean data.

We will examine and perform experiments with water data from *Waterschap Aa en Maas*. This is one of the 21 water authorities in the Netherlands. Aa en Maas can provide water data that is not available to the public, which is a major contribution to our work.

The remainder of our work is organised in the following manner. First, we define our research questions and describe our research methodology. A survey of relevant literature on both a fundamental view of outlier detection and specific outlier detection methods follows. Then we describe the available data of Aa en Maas. Afterwards, we depict the inner workings of the proposed outlier detection pipeline. Next, we describe the used modelling algorithms in more detail. This includes descriptions of how these algorithms are tailored to and to what extent the algorithms are suitable for handling the available data. Results and model evaluation follow. Afterwards, we discuss our findings and finally we conclude our work by answering the research questions and describing avenues for future research.

Chapter 2

Research questions and methodology

2.1 Research questions

We address the following research questions in this project:

1. *Which methods can be applied to detect outliers in an unlabelled, unvalidated data set of multivariate time series in a real-time setting?*

   The main research question in this project is about discovering and comparing outlier detection methods. This will be done in a real-time setting with multivariate time series data, which is a natural setting in which water authorities could use such kind of models. The data is unvalidated, which means that it is raw sensor data that has not gone through any processing steps to check or improve quality. The data is also unlabelled, which means that domain experts have not indicated whether (and if so, where) outliers occur.

2. *Can approaches based on multivariate time series improve results compared to approaches using univariate time series?*

   Intuitively, one would think that using more related (sensor) data improves performance. It is interesting to see whether this line of thought is actually true. The univariate setting might be less difficult to implement in practice, which is why it could be very interesting if this approach functions well. This setting can also be effective in areas with low sensor density.

2.2 Research methodology

For our research, we will use the data as described in Chapter 4. The data set will be divided into training (60% of the data), validation (20%) and testing (20%) sets. The training data will be used to fit different models which are described in Chapter 3. The validation data is used to perform hyper-parameter tuning. The test data is used to assess model performance. Studied outlier detection methods will not be implemented from scratch, but implementations from different software packages will be combined in an evaluation suite. Model evaluation will be performed by using synthetic outliers. Section 7.2 elaborates on this. The main idea is that we create outliers ourselves and superimpose them on the real data. Then we test how well we can detect these outliers.

Chapter 3

Literature survey

Loosely speaking, an outlier is a fundamentally deviating observation. In previous work, different authors have given different definitions of what an outlier is. Even the terminology of the term "outlier" differs. Many times, outlier detection will be called anomaly detection [10, 27, 33, 44] or event detection [36, 50], although the latter may indicate more impactful occurrences. The term which we use is a question of definition and is not crucial, but should be used consistently. We will follow Aggarwal [1] and Hawkins [16] and call this phenomenon outliers. Hawkins [16] defines the term in the following way: "An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism." This definition is very broad, which is a desirable attribute given the fact that outliers can occur on account of very different underlying reasons. Note that this definition is also open to many different kinds of water sensor abnormalities.

3.1 Fundamental view of this research problem

Before we dive any further into specific algorithms and implementation pipelines which have been previously introduced, we look at the research issue in a more fundamental way.

3.1.1 Categorisation of outlier detection

Gupta et al. [14] created a very wide overview of different categories of temporal data which each require specific techniques for outlier detection. This overview can be seen in Figure 3.1. First, we look at their categorisation and afterwards try to position our research problem into this framework. The last step could be useful to narrow down the very wide field of outlier detection to specific interesting parts for our research.

Their first and most simple category is *time series data*. In this setting, it can be relevant to look at outliers in a single time series, but detecting outlying time series (sub)sequences in a database of multiple time series could be relevant too. The second category is *data streams*. Unlike regular time series, streaming data does not have a fixed length and does not need to be finite. This needs to be approached in a different way than regular time series. For example, it could work to update some model parameters (and not refit the entire model) as new data arrives, to better capture trends in the data. The next category is *distributed data*. This category implies that multiple sensors need to work together. Each sensor has its own stream of data and the goal is to find outliers based on the joint global data. When sensor position is also important, the setting will become distributed spatio-temporal. Then, *spatio-temporal data* is considered to detect spatio-temporal outliers. These outliers consider not only the temporal neighbours (as in regular time series), but also neighbours based on sensor location. To conclude, temporal *network data* is viewed. Streams of graphs are described within this category. These

different temporal graphs can be structured differently and have different node distributions, which can change over time.

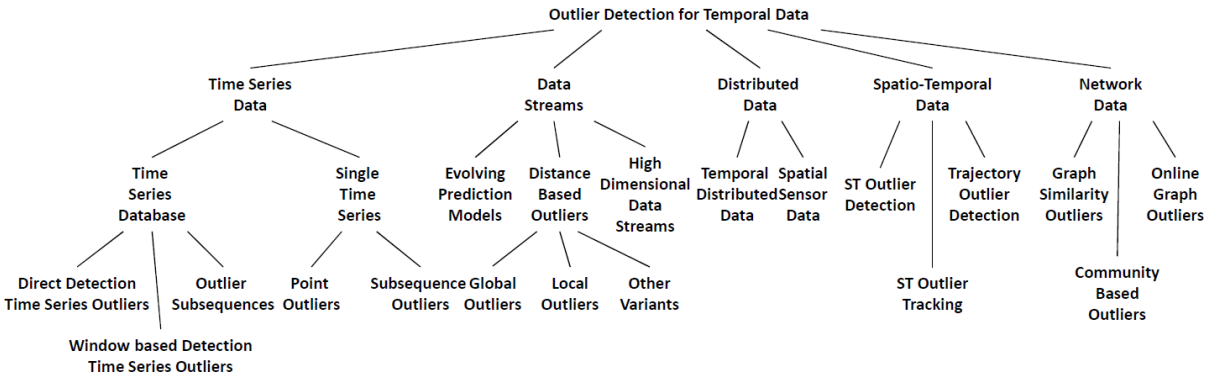


Figure 3.1: Overview of outlier detection for temporal data. This is Figure 1 from Gupta et al. [14].

When we try to position our research issue into this framework, one could say that it fits in the category of time series data. More specifically, it could be narrowed down into both the subcategories of time series database and single time series, depending on how we approach the issue. If we look at univariate cases consisting of just one time series, then the single time series approach would make sense. When considering the multivariate case, the time series database subcategory is needed, as we have multiple time series. In the single time series case, both point outliers and subsequence outliers are of interest in our research problem. In the time series database subcategory, trends which are only visible when taking multiple time series into consideration are very interesting. The multiple means of detection showed in the end nodes of this part of the overview could all be applied to our research issue. Moreover, it could be interesting to determine if a certain time series as a whole should be categorised as an outlier (in the sense of determining whether the total history of a certain sensor should be classified as an outlier). This could be useful to detect when the physical reference level of a sensor is incorrect. But, seeing how this is a whole different kind of error, it will be out of scope for our work.

This approach of bounded time series data can hold if we are interested in validating old, static data sets. However, an online streaming approach might be necessary in some use cases. More and more data will be accumulated over time, which can result in ever-changing models. These models can not always be trained on the entire data set, since the amount of data will keep on growing. However, since the data of Aa en Maas is not that big and observation intervals are not small, this approach might not be necessary. If we still consider it necessary, multiple subcategories of the data stream category, such as evolving prediction models and distance based outliers could be interesting. When considering distance based outliers, global (with respect to bounds of the whole data set) as well as local (with respect to the neighbourhood) outliers can occur and should be detected. High dimensional data streams are probably less applicable to this project, since the data will most probably not be extraordinarily high dimensional.

Since we use multiple sensors, one could think our research issue resides in the distributed data setting. However, since we are not treating the joint data of multiple sensors in the same way, this is not really the case. Joint data could be of a similar variable (e.g. water height), but could have different characteristics at different sensors. The spatio-temporal category is also not of much use. The spatial aspect is still somewhat useful, as we can look at sensors close to each other. There is some meta-information available regarding the relationships between locations, which can be used to determine useful predictors. But, we are not aiming to use location indices or x and y coordinates as predictor variables. Also, certain close by spatial neighbours are not necessarily more important than further away sensors (assuming those further away sensors still have influence). Furthermore, sensor density varies greatly over the total region, which might not work well in a spatio-temporal model. To finish this perspective, we deem the network data approach not useful, since we are not looking at graphs or structuring our situation as a graph.

3.1.2 Research issues in outlier detection

Aside from looking at a general overview of data categories, we look more formally at research issues which we can encounter. Sadik and Gruenwald [42] looked at which research issues need to be solved during the creation of an outlier detection technique for data streams. We will not look at developing extensive techniques ourselves, but existing techniques will be used and possibly combined. These research questions can improve the evaluation and judgement of these techniques. Some of the issues seem trivial and others might be problem-specific, but nevertheless, many interesting insights can be derived. A few of these are that outlier detection techniques can not hold the entire data set forever and that data points should be compared against the summary of other data points instead of the whole population. Moreover, concept drift, which is the occurrence of changes in the sensor environment and/or in trends which will change the data distribution over time, can happen and can hinder an algorithm. Gama et al. [11] describe 'real' concept drift as changes in the conditional distribution of the output given the input, while the distribution of the input may stay unchanged. They have done a broad survey on concept drift in online supervised learning scenario's. It is also stated that concept drift goes further than only the supervised scenario with labelled data. It is also interesting for unsupervised learning, but this is claimed to be a novel research topic. As such, concept drift adaptation methods fall out of scope for our project, since we are working in the unsupervised learning setting. This is the case, as we do not have labelled training examples.

Research issues in multiple streams [42] are also relevant in this project, since multiple correlating sensors can be used to detect outliers. An important issue is that correlations between streams should be monitored and data points should only be compared against correlated data points.

3.2 Building blocks in outlier detection

Before examining implemented approaches of outlier detection systems, we will identify different building blocks of these systems. Many papers propose new approaches and build on previous work, but in essence all of these papers contain similar building blocks. We will identify many of these in the following sections. There are multiple ways to approach outlier detection. One approach is to model the normal behaviour of the data, and then determine whether there is a big difference between observed and modelled normal behaviour. Another approach reverses this idea: model the abnormal behaviour and determine whether observed behaviour matches this. Most of the methods described below belong to the first category. The only building blocks we looked at which focus on describing abnormal behaviour are isolation forests (explained in Section 3.2.2) and the density-based approaches of Section 3.2.6.

3.2.1 Autoregressive models

Chatfield [5] describes many autoregressive models. These models are relatively simple ones that we might be able to use as a baseline for the more advanced models. They model the normal behaviour observed in the data. The simplest of these are univariate models, which we can extend to multivariate models. Normally, these models are used for forecasting time series, but they can be used in such a way that (user-specified) prediction intervals can be calculated as well. These prediction intervals can be used to see if an observed value is rather unlikely, and can consequently be classified as an outlier. The use of these kinds of models for outlier detection in time series is also described by Aggarwal [1].

One of the simplest of these models is the autoregressive (AR) model. An AR(p) model is defined as:

$$X_t = c + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + \epsilon_t \tag{3.1}$$

In this equation, c and ϕ_i ($i \in 1, \ldots, p$) are constant parameters and ϵ_t is white noise (generated by a random process with mean zero and variance σ^2). An AR(p) model has order p, which means that

values up to p steps back will be taken into consideration. X_t is predicted from its own past values instead of from other variables, which puts the term 'auto' into autoregression.

Another model is the moving average (MA) model. This model does not incorporate previous values of X_t, but uses prediction errors, which are alike to the previously used error term ϵ_t. A MA(q) model (with order q) is defined as:

$$X_t = c + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q} + \epsilon_t \tag{3.2}$$

Again, c and θ_i ($i \in 1, \ldots, q$) are constant parameters and ϵ_t is white noise. Combining these two models (with possibly different orders) results in the ARMA model. These models, however, assume stationary time series, whereas most time series in practice are not. To alleviate this problem, the series can be differenced. This means that instead of using X_t, a Δ-term is used, which is the difference between successive values of X. All this can be combined into the ARIMA(p, d, q) (autoregressive integrated moving average) model. The order d of integration denotes how many times the series is differenced.

These univariate models can be extended to the multivariate setting. The model names will now start with the term 'vector', denoting the multivariate setting. The equations remain more or less similar, with the difference that single coefficients and parameters now change to matrices and vectors. An example for a bivariate VAR(1) model (which still uses scalar values) is the following:

$$\begin{aligned} X_{1,t} &= c + \phi_{1,1} X_{1,t-1} + \phi_{1,2} X_{2,t-1} + \epsilon_{1,t} \\ X_{2,t} &= c + \phi_{2,1} X_{1,t-1} + \phi_{2,2} X_{2,t-1} + \epsilon_{2,t} \end{aligned} \tag{3.3}$$

It can be generalised to the VAR(p) model, consisting of K variables and using vector notation:

$$\mathbf{X}_t = \mathbf{c} + \mathbf{A}_1 \mathbf{X}_{t-1} + \mathbf{A}_2 \mathbf{X}_{t-2} + \ldots + \mathbf{A}_p \mathbf{X}_{t-p} + \boldsymbol{\epsilon}_t \tag{3.4}$$

where $\mathbf{X}_t$ is a $K \times 1$ vector of variables, $\mathbf{c}$ a $K \times 1$ vector of parameters, $\mathbf{A}_i$ (i $\in 1 \ldots p$) are matrices of size $K \times K$ and $\boldsymbol{\epsilon}_t$ are $K \times 1$ error values, which are defined similarly to ϵ_t in Equation 3.1.

Lütkepohl [31] describes how to compute prediction intervals when forecasting VAR processes. For Gaussian VAR processes, the prediction errors are (multivariately) normal distributed and can be calculated with relative ease [30]. When describing non-Gaussian time series with unknown distribution, other methods, like bootstrapping methods, may be needed to calculate the prediction intervals.

3.2.2 Isolation Forests

Isolation forests (IF) is an interesting tree-based technique [29]. As the name suggests, ensembles (forests) of trees are built to detect outliers in data sets. The approach focuses on the concept of isolation, instead of constructing a profile of normal data instances. When building a tree, a random split between minimum and maximum feature value on a random feature is performed. Then, all data points are on one side of the split, and further splitting on the two subsets ensues. When testing data arrives, it is calculated how many splits are necessary to isolate that data point from other data. Since outliers are rare by definition, they will need relatively few splits, whereas inliers need many splits to isolate them from other data. Since the tree building process happens in a random fashion, results from multiple trees are averaged.

IF overcome issues like swamping (wrongly identifying normal instances as outliers) and masking (the existence of too many outliers concealing their own presence). This is done by sub-sampling, which controls data size, to better isolate examples of outliers. Also, every isolation tree can be specialised due to the aforementioned sub-sampling. Liu et al. [29] compare the algorithm against other methods like Local Outlier Factor (LOF) and Random Forests (RF). IF outperforms these other algorithms

with respect to AUC as well as computing time. These properties of accuracy and efficiency make IF worthwhile for high volume databases in real-life applications. IF works well both in training data where outliers occur commonly as in outlier-free data.

IF is not without issues, however. Just like distance-based methods, it suffers from the curse of dimensionality when using high dimensional data. This could be overcome by using attribute selectors (which can select variables to use based on attributes like kurtosis, which measures 'peakness') to reduce dimensionality of the data sets. Still, this might pose some problems in the data sets when using large numbers of relevant sensors.

IF has been implemented for streaming data by Ding and Fei [10]. They implemented a sliding window approach to ensure that the model can handle infinite amounts of streaming data, as storing and scanning the whole data set will not be feasible. A method to handle concept drift was also given. To do this, a rough estimation of the outlier rate is necessary. This rate will be used as a threshold after scoring and ranking (batches of) input data. If the observed outlier rate is higher than the threshold, concept drift is detected and the model is retrained on a more recent sliding window. To set this parameter, prior or domain knowledge about the data should be known. This may be problematic in real-life approaches. Another issue is the width of the sliding window. According to Ding and Fei [10], the most common method in literature is to use trial and error to set the width. The width is fixed manually in this approach, but an automatic or varying self-adaptive width could improve results.

3.2.3 Long Short Term Memory (LSTM)

The LSTM model is a specific recurrent neural network (RNN) class, which has a more complicated structure to handle retaining information in a better way. It was invented in 1995, with the original paper about it published in 1997 by Hochreiter and Schmidhuber [18]. This model was invented to overcome the vanishing gradients problem often found when training RNN models. LSTM's have a widespread use and are also applied in outlier detection. Like an AR model, they are able to model the normal behaviour of the data. LSTM's can be really useful for outlier detection, since they do not need time windows as they can learn long term correlations [33]. As a result, complex multivariate sequences can be modelled accurately. However, this might not necessarily hold if we take concept drift into account.

Jozefowicz et al. [19] make an empirical comparison of multiple RNN architectures to find out if the LSTM model can be improved upon. It is concluded that LSTM's perform really well, but are not the best models for every kind of problem specification. One recent and promising alternative is the Gated Recurrent Unit (GRU), introduced by Cho et al. [7]. This is a similar but somewhat simpler approach than the LSTM and could be useful for us as well.

Malhotra et al. [33] use an architecture of stacked LSTM's. This network is trained on normal, outlier-free behaviour. To detect outliers, the model predicts multiple time steps into the future. When the actual value is observed, multiple error values can be calculated by using all these predictions. A multivariate Gaussian distribution is fitted on the error values. The likelihood of the observed value on this distribution is compared against a threshold (which is calculated by maximising the F-score on a validation set), to determine whether it is an outlier or not. Is it shown that the stacked LSTM network is suited to model normal time series behaviour. This might prove problematic in our case, since outlier-free training data cannot be guaranteed. Also, short-term as well as long-term dependencies can be learned by this model. The model is shown to have considerably low recall, but it is claimed this is the case because not all data points in an anomalous section will be classified as outliers. Instead, only some data points are classified as such which may suffice to detect that deviating behaviour has occurred. Another problematic issue that might arise is that this model does not focus on streaming data, which may be necessary in our case.

Laptev et al. [26] focus more on the forecasting problem. Although this is different from our approach, it might still be relevant if we can determine whether the forecasted values are actually outliers. A multi-layer LSTM model is used as an autoencoder to be able to extract features. These are then plugged

into a LSTM forecaster. Before all this is done, data is first normalised and de-trended, which might be interesting for us as well.

There are some methods that can be used for calculating prediction intervals when using neural networks. Section 3.2.4 goes into more detail in how these intervals can be used. Zhu and Laptev [53] explain a method where dropout is used. Dropout is a neural network technique where nodes can be "dropped out" and removed from the network, normally during the training phase. Afterwards, the network is kept intact. This is a technique invented to reduce overfitting, but also works well to provide an uncertainty estimation when used during the test phase.

Another well known method to calculate these intervals is bootstrapping [20, 26]. By using this method, an ensemble of neural network models is needed, similar to how vanilla random forests use ensembles for classification. Like using dropout, this can not only be used to decrease overfitting, but also to turn the multiple estimations of the ensemble into prediction intervals. The downside of this approach, however, is that large ensembles of neural networks can require a long training time.

3.2.4 Quantile regression

Quantiles, or percentiles, are the general case of for example the median and quartiles, and divide the population in segments. Quantile regression is relevant to our research problem, since values calculated for extreme quantiles can be used as thresholds to classify data as outliers or not. As explained by Koenker and Hallock [21], quantile regression extends the ideas of quantiles to the estimation of conditional quantile function-models. In these models, quantiles of the conditional distribution of the response variable are expressed as functions of observed predictor variables. In other words, instead of just inferring the conditional mean of a response variable, we are able to estimate the conditional distribution by using quantile regression. Quantile regression can describe the normal behaviour seen in the data. Extreme quantiles can be viewed as the boundary between normal and outlier data.

Quantile regression can be implemented in multiple ways. One way to do this is by using the quantile regression forests algorithm introduced by Meinshausen [35]. This technique builds upon the well known random forests algorithm, which is suitable for regression problems. The fitting procedure is more or less the same as in regular random forests, except that all observations are saved in nodes, instead of only the mean. During the prediction phase, the observations are dropped down all the trees and weights of each observation from every tree are computed and then averaged. Finally, these weights are used to compute the estimate of the distribution function.

By using this method, prediction intervals can be built to detect outliers in the data. The width of the prediction intervals is not fixed, but depends on the observed data. A downside of this method seems to be that it is relatively slow and does not scale that well to bigger data sets.

A much different way to calculate these quantiles is by using neural networks. Different quantiles can be set as output nodes by using specific loss functions, as shown by Rodrigues and Pereira [41]. In this approach, a tilted loss function (also known as pinball loss) is used, as defined in [21]. By using this function, multiple quantiles as well as the mean can be calculated by propagating the predictor variables through the neural network. It is shown that this approach works in simpler univariate time series forecasting problems and in more complex spatio-temporal models too. In the more complex models, convolutional LSTM's [51] are used to address redundancy for spatial data, which is normally encountered in vanilla LSTM's. Another beneficial effect of this method is that the quantile crossings problem, which occurs when quantiles overlap, is alleviated. This crossings problems occurs when time series of different quantile values are crossing, instead of having one series consistently higher than another series. The calculated quantiles can be used as thresholds for the outlier detection problem. We are also able to use this method and possibly extend it to the method using convolutional LSTM's, although the setting of the data of Rodrigues and Pereira [41] is different than ours, since the grid-like spatio-temporal aggregation they performed might not be possible for us.

3.2.5 Autoencoders

The autoencoder is a specific neural network architecture, which was originally invented to be able to perform principal component analysis (PCA) in a non-linear way by Kramer [23]. PCA can be used to perform dimensionality reduction in a data set, which might remove insignificant dimensions of the data. Autoencoders can extend this approach to a non-linear fashion, which can be achieved by involving deep neural networks. Autoencoders work by compressing the input to a much smaller dimensional latent space with user-specified dimensions and then trying to reconstruct the original signal. This means that the number of output dimensions will be the same as the number of input dimensions. The latent space representation is a compressed view of the original data and can be used as a feature extraction method, as shown in practice by Laptev et al. [26] and described more generally by Aggarwal [1].

Aside from applying autoencoders to dimension reduction, it can also be used for outlier detection, as explained conceptually by Aggarwal [1] and shown in real-world sensor data by Reunanen et al. [40]. The neural network architecture used is identical to the architecture used when performing dimensionality reduction. A key difference, however, is that the output of the autoencoder is now vital, instead of the latent space. The concept behind using autoencoders in this way, is that the network should be able to learn what normal, or outlier-free data is. This is based on the assumption that the network is fed outlier-free data. When it tries to reconstruct such data, it should be able to do well and create a reconstruction which differs not too much from the original data. The reconstruction error, which is the squared difference between the reconstruction and the original data, is expected to be low. When the network is fed outlying data, the network should not be able to reconstruct this well, as it did not encounter such data during the training phase. As a result, the reconstruction error will be high. So, by examining the reconstruction error, one might be able to detect outliers.

Generally, this technique is applied on data sets of machines and of manufacturing processes like the data set of Ranjan et al. [39]. High-dimensional data of one process or machine can be inputted in the network and then reconstructed. As a result, reconstruction error can be used to define when a process or machine malfunction has happened.

3.2.6 Distance-based methods: Nearest neighbour and density-based approaches

There are many different kinds of methods which attempt to capture regular patterns in data by using clustering or density-based methods. The idea behind using the concept of clustering for outlier detection is that inliers are assumed to be somewhat clustered. So, observations which are far from (one of) the main clusters could be said to be outlying.

Pimentel et al. [37] and Talagala et al. [43] describe many different methods based on this concept. These methods are fundamentally different than regression-based methods. In regression-based outlier detection methods, we will claim that an observed value is an outlier because its predicted value is too far off from the actually observed value. Here, we will not work with predicted values, but with data distributions instead. A possible downside, however, is that "correct" values can not be estimated with these methods if we see that a certain value is an outlier. We need to resort to regression-based methods or other kinds of imputation for that. Moreover, one could argue that by using this method, the temporal aspect of time series gets lost. If, for example, a sensor has a regular error that occurs every once in a while, all these observations could be clustered together, making it hard to actually find that these values are outliers. The use of lag features could alleviate this issue to some extent. Nevertheless, Talagala et al. [43] and Leigh et al. [27] managed to get decent results with these kinds of techniques applied on time series.

Some of the described methods in [43] are based on k-nearest neighbour distances. They operate under the assumption that outliers are isolated in data space. As a result, points with the largest k-nearest neighbour distances can be said to be outliers. One of these methods is the HDoutliers algorithm. This unsupervised outlier detection algorithm looks for outliers in high dimensional data and assumes that

there is a large distance between outliers and regular data. This method utilises nearest neighbour (Euclidean) distances between data points. The data is normalised first to prevent variables with high variance to have a disproportional high influence on the calculations. A big amount of data can be handled by using the leader algorithm, which forms several clusters. A typical member is selected from each cluster and nearest neighbour distances are calculated. A calculated threshold is then used to determine whether a data point is an outlier or not. The calculation of this threshold is based on extreme value theory.

An improvement on the HDoutliers algorithm was presented by Talagala et al. [44]. This algorithm called "stray" solves problems such as only using nearest neighbour distance, encountered issues due to clustering via the leader algorithm and issues with threshold calculation. These two methods look promising, but have not seen that much use yet in practice. Other alike methods are aggregated k-nearest neighbour distance (KNN-AGG) and sum of distance of k-nearest neighbours (KNN-SUM). A difference between these two methods and HDoutliers is that they are robust to outliers with outlying neighbours. If two outliers are near to each other, HDoutliers will not detect them. This is the masking issue, just as described in 3.2.2. That might make HDoutliers inappropriate for our setting. These two KNN algorithms take k points into account and behave similarly. Their difference lies in a different weighting of these points: KNN-SUM calculates the sum of distances to these points whereas KNN-AGG calculates a weighted sum with nearer neighbours having a higher weight.

Other methods can be density-based. They calculate outliers based on the isolation of points compared to surrounding neighbours. Then, observations with low densities can be labelled as outliers. These approaches are not extremely different, but have a subtle distinction. This approach is more or less the same as used in isolation forests in 3.2.2. One of the most well known of these kind of algorithms is Local Outlier Factor (LOF) [37, 43]. LOF calculates outlier scores based on isolation compared to surrounding neighbours. Points with lower density than their neighbourhood are classified as outliers. This density is computed by looking at the average reachability distance of k nearest neighbours. This algorithm is also affected by the masking issue, however. Other algorithms, like Connectivity-based Outlier Factor (COF), try to alleviate this [43].

3.2.7 One-class SVM

Support vector machines (SVM's) are a well known method in data science. Traditionally, SVM's were used to classify data into two classes. This is done by calculating an optimal separating hyperplane, which maximises the margin of the classifier. Less formally, this indicates that the hyperplane is placed in such a way that the distance to the nearest training samples is maximised. The goal of the maximisation is to improve the confidence in the predictions.

Aside from general classification, SVM's can also be used for outlier detection. This can be done by using one-class SVM's (OC-SVM's) to separate data of one specific class from all the other data [37]. Applying this to outlier detection, all the regular data is modelled in one class, and the rest will be outliers. Although most research focus seems not to be on time series data, some uses of SVM in time series data have been seen [25, 32]. However, there are some major issues in outlier detection by using OC-SVM's. Aggarwal [1] and Lamrini et al. [25] state that OC-SVM's are only trained on the positive class, or otherwise put that there are only examples from a normal class instead of outliers as well. We can not possibly assume that the training data is outlier free. This makes it not an apt approach for us.

If we do have outliers in the training data, the SVM will be very sensitive to these outliers, as outliers can have much effect on the decision boundary. This is a characteristic which may be very detrimental for us. Some solutions of this issue have been introduced by Amer et al. [2]. Nevertheless, these approaches are not readily available in code, it is unsure whether this also works for time series and the issues of sensitivity to outliers may still remain. To conclude, SVM's will not be pursued in our research.

3.3 Implemented approaches of real-life outlier detection systems

In previous work, many different approaches for handling outlier detection in online time series data have been introduced. For example, in Leigh et al. [27], a ten-step outlier detection framework for high frequency water-quality data was introduced. This framework states the importance of collaborating with end-users to improve model and recommendation quality. They suggest using multiple kinds of outlier detection algorithms to provide optimal performance. Feature-based methods like KNN-based methods and HDoutliers as well as (auto)regression based methods such as ARIMA are used. They do, however, assume that a training set without outliers is available (or can be made available by validation by domain experts), which is a bold assumption. In their study this worked, since very few sensors were available. In a setting with a relatively high sensor density, like in our case, this is probably not feasible. Moreover, because of the low sensor density here, it was not possible to use sensors values to make predictions for other sensors values. This is possible in our data set, which may require a different approach. Furthermore, they define outliers as faulty data due to a technical error in a sensor. In our approach, we might encounter other cases of outliers (such as sludge in front of sensors or the presence of beaver dams) as well.

Talagala et al. [43] performed similar research on the same data set, where HDoutliers as well as KNN-based methods were used. They seem to have outperformed density-based algorithms such as LOF, COF, Influenced Outlierness and Robust Kernel-based Outlier Factor. To achieve decent results, they required carefully selected data transformation methods to adjust their data. This was due to only seeing a clear separation between outliers and typical data points after these transformations had been performed.

Reunanen et al. [40] presented another idea. They described a system to detect as well as predict outliers in streams of sensor data. The outlier detection relies on an autoencoder with some extra parameters to adapt to an evolving inlier data distribution. It also utilises a count-based sliding window. The reconstruction cost of new data is used to determine whether it should be classified as an outlier. The system does not need outlier-free training data or labelled training examples. Outlier prediction uses logistic regression, stochastic gradient descent and also the hidden representation that is outputted by the autoencoder.

A process for water data measured by ultrasonic sensors was introduced by Bae and Ji [3]. In this work, ultrasonic sensors detect water levels at a really small time interval (2.5s). This is different from our data set, where measurements are made every 15 minutes or hourly. Their system uses median absolute deviation (MAD), exponentially weighted moving average (EWMA) and can automatically detect the best windowing size. It does need some user-specified parameters, such as the initial cutoff range, the minimum and maximum window size, the rejection criterion β, a smoothing constant factor and sensor resolution. First, they use domain information to determine an initial cutoff range, which can detect the majority of the outliers. Then, by using MAD, modified Z-scores and a cutoff range β, outliers are detected and removed from the data set. Afterwards, the data is smoothed by using the EWMA. Their approach seems valid, although it is unsure if it generalises to lower frequency water height data with much fewer outliers. In their data many outliers are present due to their ultrasonic way of measuring water heights as well as due to occurring water waves.

A more general water sensor platform was introduced by Whittle et al. [50]. This platform manages and analyses sensor data for many different ends, such as water demand prediction. More relevant to our research, online event detection for events such as pipe bursts or leaks is also implemented. The system incorporates a lot of domain knowledge, including a real-time hydraulic model of a water distribution system. Event detection is among others based on wavelet decomposition. In this way, suddenly changing pressure levels can be detected. Another used technique is time-domain statistical analysis. By doing that, fast and large changes can be detected, after filtering noise. Although this system is applied in practice in Singapore, it is of less use to us, since it is not focused on modelling available data, but more

on domain knowledge models. The detection is also not explained well enough to be replicable.

Perelman et al. [36] created an event detection system in multivariate water quality data by using ANN's and Bayesian sequential analysis. ANN's (relatively simple MLP's in this case) were trained on the training data, after which the residuals were calculated. Based on these residuals, a bandwidth was created so that 96-99% of the residuals are in this range. When classifying new data, these bandwidths (which exist per predictor variable) are used to determine if a data point is an outlier for a certain variable. Then, probability of events are updated by using sequential Bayesian analysis. Eventually, actual events are declared when multiple of these probabilities exceed some user-specified threshold value. For our work, the Bayesian step might not be needed, since we are not looking at system-wide outliers. What is also interesting in this research, is that water contamination events were simulated in the data to ensure a sufficient number of outliers. This could also be applicable to our work.

As previously stated, Zhu and Laptev [53] used dropout to calculate prediction intervals. More specifically, a deep neural network was used to calculate confident predictions on univariate data. Their proposed model described prediction uncertainty as a combination of model uncertainty, model misspecification and inherent noise. They tried to capture the misspecification by using a LSTM encoder-decoder which enables representative feature extraction in a later stage. Model uncertainty was captured by using dropout and the noise was estimated by using an independent validation set. This model was used in practice for outlier detection at Uber. When observations fell outside the 95% prediction interval, they were marked as outliers. This model was shown to be scalable and resulted in high performance.

Hill and Minsker [17] created an outlier detection system for univariate wind speed sensor time series data. They used univariate autoregression methods and then calculated prediction intervals. Naive prediction (also known as the persistence model), nearest cluster (a modification of KNN), single-layer linear network and MLP were compared. For all these methods, prediction intervals were calculated by using mean and variance which were estimated on 10-fold cross-validation. Cleaning of training data was done by using the naive prediction. A downside of this approach is that the interval width is fixed until models are retrained. Varying interval widths (as in Section 3.2.4) can be useful in outlier detection, since knowledge about the certainty of predictions can be helpful in the classification of possible outlying behaviour. Also, models need to be retrained relatively often to update these intervals. Interestingly, their naive predictor model, which is a very simple model which predicts the last observed value, actually worked well. This naive predictor method might be useful for us as a baseline model to compare more complex models against.

Chapter 4

Data overview

In our research, we analyse data from a Dutch water authority (Aa en Maas). Their sensor data is either measured every hour or every 15 minutes. A measurement frequency of 15 minutes will be assumed unless specified otherwise. We use this high frequency data, as it will lead to more data overall and to the possibility for model improvements, as models might learn more detailed patterns. Multiple variables measured by sensors can be queried. For example, we can query water heights on the upper and lower part of a weir as well as water flow rates. Combining these variables leads to multivariate data sets.

We chose 4 weir locations for the analysis and model evaluation. These weir locations are 102BFS, 103HOE, 104OYE and 201D and are shown in Figure 4.1. At these locations, we have access to water height on the upper part and the lower part of the weir, and also to water flow rate and weir shutter height. We used water heights on the *upper* part of the weirs for the analysis, which will be indicated in many of our figures as *_boven*. The advantages of these specific combinations is that domain experts have flagged this data for possible outliers. These sensors have no flagged outliers in them, which makes the analysis in Chapter 7 more straightforward. We did not use multiple variables of these sensors, as this did not improve performance. Section 5.2 will go into more detail on this topic.

We gathered all data between 05-06-2015 and 01-07-2019. According to the 60% - 20% - 20% split explained in Section 2.2, this means that the training data is in the range of 05-06-2015 - 13-11-2017, the validation data ends at 06-09-2018 and the testing data ends at 01-07-2019. Some coarse pre-processing steps were applied before we retrieved the data. Namely, flatlines are non-existent in water height data, as values are not allowed to be the same for a period of longer than 3 hours. Also, hard-coded rules for detecting impossible and out-of-range sensor values were already applied previously on most sensors, which removed the need to implement sensor-specific rules ourselves.

4.1 Weir connectivity

Our research will focus on water data from weirs. To get a better sense of what we are dealing with, we show images of the weirs used in the analysis in Figure 4.1.

It is interesting to know how weirs are connected via waterways. This might give use useful information in deciding at what sensor sets we want to look and can be helpful when determining if models make any sense. In Figure 4.2, we see a zoom of an area with many weirs. Aside from interesting automatic weirs, manual weirs also exist. We notice that there are many more weirs in total (Figure 4.2a) than there are automatic weirs (Figure 4.2b). Nevertheless, we can still notice some automatic ones relatively close to each other, which could be beneficial to the models. This is the case, as we would expect the time series of the geographically close sensors to be correlated. This insight eventually led us to a method of sensor selection by examining their correlation. Section 5.2 will explain this in detail.

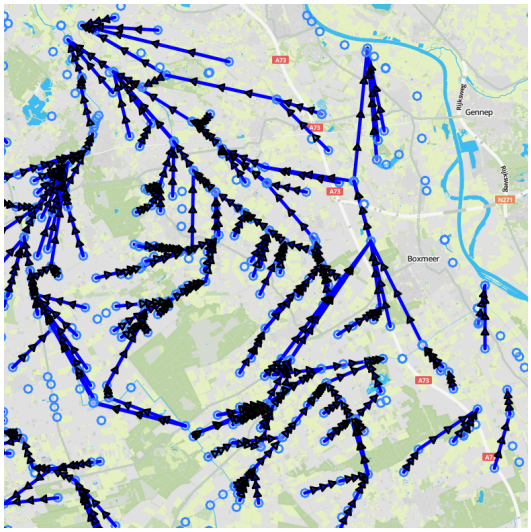
(a) Weir 102BFS, viewed from the lower part of the weir.

(b) Weir 103HOE, viewed from the upper part of the weir.

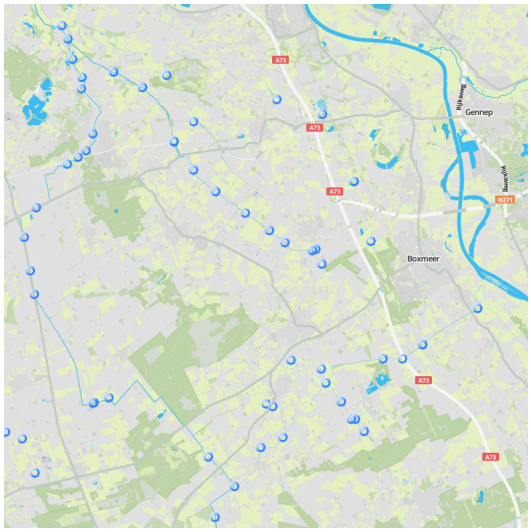
(c) Weir 104OYE, viewed from the upper part of the weir.

(d) 201D, viewed from the lower part of the weir.

Figure 4.1: The 4 weirs used throughout our work. Images taken from Aa en Maas.



(a) Zoom in on a specific view, showing all weirs in the area, with connectivity information (where available) being used to draw arrows in the direction of the waterway.



(b) Zoom in on the same view, but now only showing all automatically controlled weirs and excluding all manual ones.

Figure 4.2: Weir locations and connectivity. The automatic weirs of Figure 4.2b transmit high frequency sensor data and are the kind of weirs that are used throughout our research.

4.2 Time series examples

An example of water height data on the upper part of a specific set of weirs is shown in Figure 4.3.

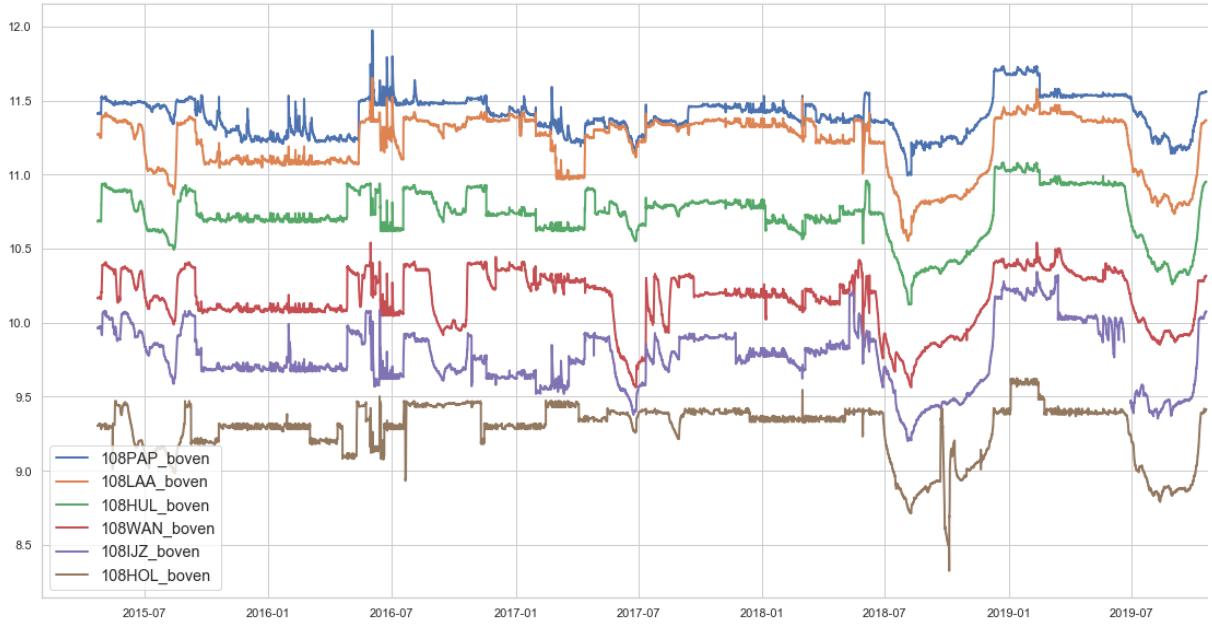


Figure 4.3: An example of available water height data. These six sensors are all on the same body of water and are relatively close to each other (about 6km as biggest distance).

When looking at this data, we see that it is not without errors. For example, the "108HOL_boven" time series (bottom line) has a very strange swing (with respect to the other sensors) around October 2018. When looking at this same series, some other minor spikes can also be encountered. Furthermore, missing values can occur, as seen in the "108IJZ_boven" time series at the right. This data set has relatively few missing values (approximately 2500); other sensor sets have more.

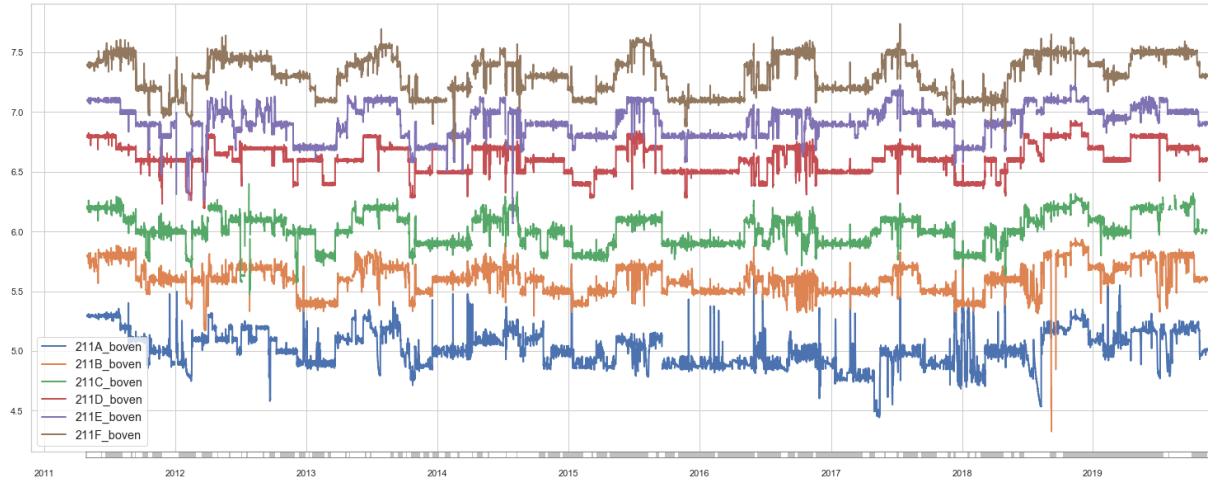


Figure 4.4: Another example of available water height data. Again, these six sensors are all on the same body of water and are relatively close to each other (about 7 km as biggest distance). The bar at the bottom indicates whether all time series have observed values at that point in time. If that is the case, the bar will be filled grey at that time step. As can be seen, there are many time periods where there is missing data.

One of those sensor sets with more missing values is visible in Figure 4.4. This figure describes water height values gathered by 6 sensors, but also shows a bar at the bottom of the figure signalling whether all time series have data at that point. As we can derive from this picture, it is probably not doable to only look at time periods of sensor sets where no missing values occur. Thus, we must find a smart way to deal with this missing data. Imputation of missing data is a possibility. Also, many easily discernible outliers are present in this data. Many sudden jumps of a few time steps are present. Moreover, it sometimes occurs that the water level of a certain sensor is higher than the water level of another sensor located upstream, which is physically impossible.

Some quick initial data analysis has now shown that not all sensor sets are of equal data quality. The data of Figure 4.3 is relatively high quality, for example. However, when looking at other sensor sets, like Figure 4.4, these time series seem more noisy. In addition, there are many time intervals at which data is missing for one or more sensors, as shown by a white value in the bar below the graph. Some sensors might not have any data for a significant period of time. Thus, it is interesting to experiment with multiple sensor sets of different qualities, to see how data quality affects the results.

Chapter 5

Outlier detection pipeline

To go from raw data to outlier classification output, many different steps need to be performed. This sequential procedure will be called the outlier detection pipeline. The pipeline will be specified and thoroughly explained at every step. We will also explain why we have or have not used specific approaches and have or have not explored interesting avenues along the way. The proposed pipeline consists of the following steps:

1. Data gathering
2. Sensor selection
3. Imputation
4. Feature engineering
5. Feature scaling
6. Modelling (normal) behaviour
7. Predicting behaviour
8. Outlier classification

It is interesting to note that we did not implement pre-processing of training data in the pipeline. One could argue for applying this step to improve the data, as we have seen in Chapter 4 that the data has its flaws. A downside of these flaws is that we may train the models on flawed data, which might be detrimental when modelling normal behaviour of the data. Leigh et al. [27] detected impossible and out-of-range sensor values with automated, hard-coded rules and classified them as outliers. We could apply this idea to the training data. A downside, however, is that these rules are highly sensor-specific. Other pre-processing avenues might be detecting flatlines and jumps in the data. Extreme values could be removed automatically as well.

We have to be very cautious when applying these methods. It is vital to do this carefully, since we do not want to discard useful data. Also, these pre-processing methods could require different parameter values for different kinds of water variables. For example, a water height time series could have very different characteristics than a water flow rate time series.

Due to the aforementioned previously applied database pre-processing steps, flatline removal is not necessary in water height data. Moreover, flatlines are considered normal in water flow rate data, as water flow in some minor water passageways can easily be 0 during the summer. A method for removing more subtle jumps and extreme values in the training data was experimented with. Some data exploration gave us the insight that many sensor value movements that can be considered as extreme values or jumps can actually be normal behaviour. If the behaviour of weirs is adjusted, water height

can rise or fall very quickly. When examining sensor time series, we found that multiple sensors can have these movements in the same, highly correlated way. There are also extremes values which do not fall under this category. But, when removing these, we have a high chance to also remove the previously explained normal behaviour. Since this is an extremely sensitive issue, we decided against implementing pre-processing of data in the pipeline.

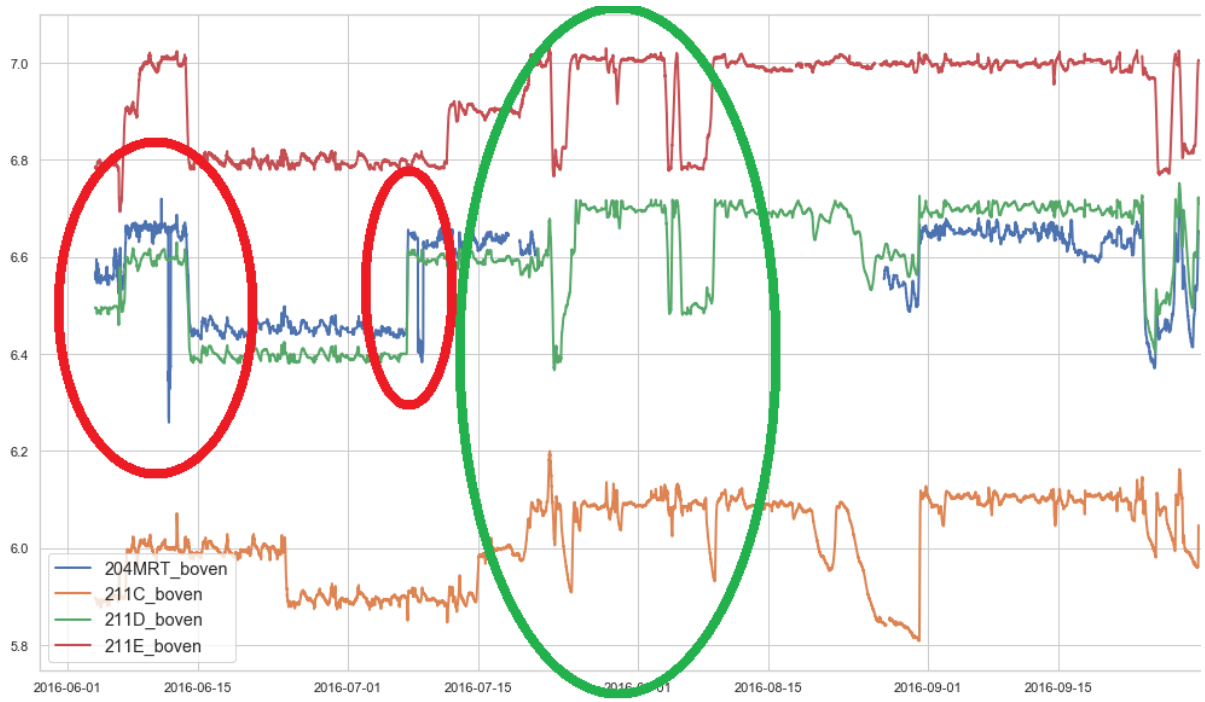


Figure 5.1: Cleaning of training data is a sensitive issue. Outliers circled in red should be removed, but green circled values should definitely remain.

5.1 Data gathering

It was fairly easy to access vast quantities of water data. We could access water height on the lower and upper part of the weir, weir height and water flow rates of the sensors. We could collect data of multiple sensors at once. Multiple variable kinds can be combined in one data set, so that the entire set of information of the automatic sensors is available. After this step, time series of all desired sensors and variables are available and can be used in subsequent pipeline steps.

5.2 Sensor selection

Since Aa en Maas has many different sensors, we may retrieve a large data set of which much data might be irrelevant for predicting outliers of one specific time series. This does not apply to univariate modelling; it is only relevant to the multivariate case. Moreover, having many irrelevant time series incorporated in the data set might lead to the inability of models to detect actual and useful relations in the data. This may worsen model performance and also model run-time.

It is sensible to perform the sensor selection at this stage of the pipeline instead of combining it with the feature engineering described in Section 5.4, because the amount of data may decrease significantly after this step. This is important for a run-time acceleration of the imputation step of Section 5.3.

It is possible to use multiple variable types of a certain sensor. For example, we might use a combined data set of water flow rate, weir shutter height and water height on the upper and lower part of the weir.

This can lead to a vast amount of data. After some initial experiments in testing whether using multiple variable types helps performance, it turned out that adding multiple variable types of a certain sensor could actually be detrimental. Domain experts from Aa en Maas explained to us that there is a very specific relation between weir shutter height and water height, for example. In some time series, this relation is nicely captured in a certain time span. However, it often happens that this relation only holds for a short period of time, after which the (cor)relation falls of, which worsens modelling performance. If we force a model to incorporate this information, then it can happen that the predictions of a model actually look less likely and realistic.

We removed all sensor time series with too many missing values. For time series with a low or moderate number of missing values, we will present a solution in Section 5.3. The rationale behind removing time series with too many missing values is that the models may not be able to learn the actual relations between the different series well enough and thus may have a negative effect in the modelling. Also, performing imputation on these series might be problematic, both in regard to run-time as to accurate reconstruction. For our aims, a balance between keeping enough relevant information and discarding non-sensical information is vital. Therefore, we decided to discard all time series with at least 10% of their values missing. This value seemed to work well for the time series and still keeps the possibility of keeping series with relatively many missing values.

To get a sense of the usefulness of the now remaining sensors, we examined the correlation between different series. We only kept time series which were (highly) correlated with the time series in which we wanted to detect outliers. It was a possibility to impose a threshold on these correlation values for sensor selection. However, this could lead to a necessity of a variable threshold, since correlation values can differ to large extent. Also, it could then happen that some predictions take different numbers of other sensors into account than others. This hinders the model hyper-parameter decision process and the feature engineering step. We did not opt for a correlation threshold. Instead, we kept the 4 most correlated sensors to the sensor of interest to circumvent these issues. Nevertheless, it is the case that the value of the lowest correlated sensor of these 4 can differ strongly among different sensors.

As an addition to this procedure, we experimented with an approach which not only examines correlations between two time series, but also correlations with one of the series shifted back in the past. This could be relevant, since it might be possible that time series are more highly correlated if one of them is shifted. This makes sense intuitively, since the stream of water is not instant and the same water needs some time to pass by sensors further downstream. Although this first seemed to be an interesting addition, it turned out that it does not really improve the selection. When we looked at shifted series (shifted by 1, 2, 4, 6 and 12 hours, by 1 and 2 days and by 1 week), we noticed that the maximum correlation of a sensor against all those shifted series did not differ substantially from the correlation of the sensor against the original series. The correlation values varied a little bit, but this was not nearly enough to impact the sensor selection. As a result, we decided against the addition of using shifted correlations in the selection procedure.

The correlation selection step is an example of applying the filter method for feature selection. Filter methods analyse properties of available data to limit the number of features and do not make any assumptions about models used further on in the process, as noted by Hall [15]. A different method of feature selection is the wrapper method, as explained by Kohavi and John [22]. This method wraps the feature selection around the to be used modelling method. An optimal feature subset is tailored to a specific algorithm and can use cross-validation accuracy to decide on removing or adding a specific feature. An advantage of applying this approach to our case, is that smart decisions can be made regarding mutually correlated features. The correlation selection will result in a set of time series which are highly correlated to the time series that we are modelling, which is desired. However, the multiple time series in this feature set might be too alike to one another and it is possible that they do not all contribute to the modelling. A wrapper approach might add one feature in a forward selection sense, and then see that another feature will not have any added contribution (meaning that the addition of this feature does not increase cross-validation accuracy) and then decide not to add that one.

A study of a combined filter and wrapper approach for neural networks has been performed by Crone and Kourentzes [8]. They note that wrapper approaches are often inefficient and are computationally expensive because of the large number of possible feature combinations. Since running and evaluating all the different models will probably already be quite computationally expensive, we consider using a wrapper method and performing related experiments out of scope.

5.3 Imputation

Just like in most real-life scenarios, it was often the case that some data points were missing in the available data. In our research setting where we are dealing with many different sensors, missing data can have many causes. Examples of such causes are network or connection issues, sensor malfunctioning and a switched-off logging computer. Many models will have serious issues when confronted with missing data. Some models are even unable to deal with this at all. Thus, we needed an alleviation for this issue.

Missing data can be handled in multiple ways. For example, we can discard all values of multiple time series at a certain time step, if one of the time series has missing data at that time. For us, however, this is undesirable, as we would like consistency between successive time steps. If we have large gaps in the data and we remove all these missing time steps, we might see spurious relations in data which do not correspond with reality. For example, it is possible that three days are skipped after removing gaps. Two successive values can now have a time gap of three days, whereas this normally is 15 minutes. This might lead to false assumptions and erroneous models.

Another way to deal with missing data is by using imputation. This means that we fill in the missing data. There are many ways to perform imputation. For instance, it can be done simplistically by using the mean value of a certain sensor for all missing values. More complicated methods, which might follow the trends in the data better, have also been invented. Since we are often modelling relations by using time series of multiple sensors, we might be able to exploit the correlation between these sensors in imputation methods. When a sensor value is missing, it might be sensible to try and use available values of other sensors to help in the reconstruction of the missing value.

We created an imputation benchmark to compare different imputation methods and decide which method we wanted to use.

5.3.1 Imputation benchmark: A comparison of imputation methods

When filling in missing values, it is hard to quantify how well this has been done. A qualitative visual inspection is possible, but we desire quantified and objective measures for comparison between different methods. To be able to see if the used imputation functions perform decently, we created a benchmark. We used data sets without missing values and artificially created gaps in these sets to see how well different imputation methods can recreate the missing values.

We created gaps of different widths. In order to create a realistic benchmark, we used widths that were observed in other data sets where gaps were present. Moreover, some really large gaps were sometimes added to check if the models can handle them as well, as they were also present in some data sets.

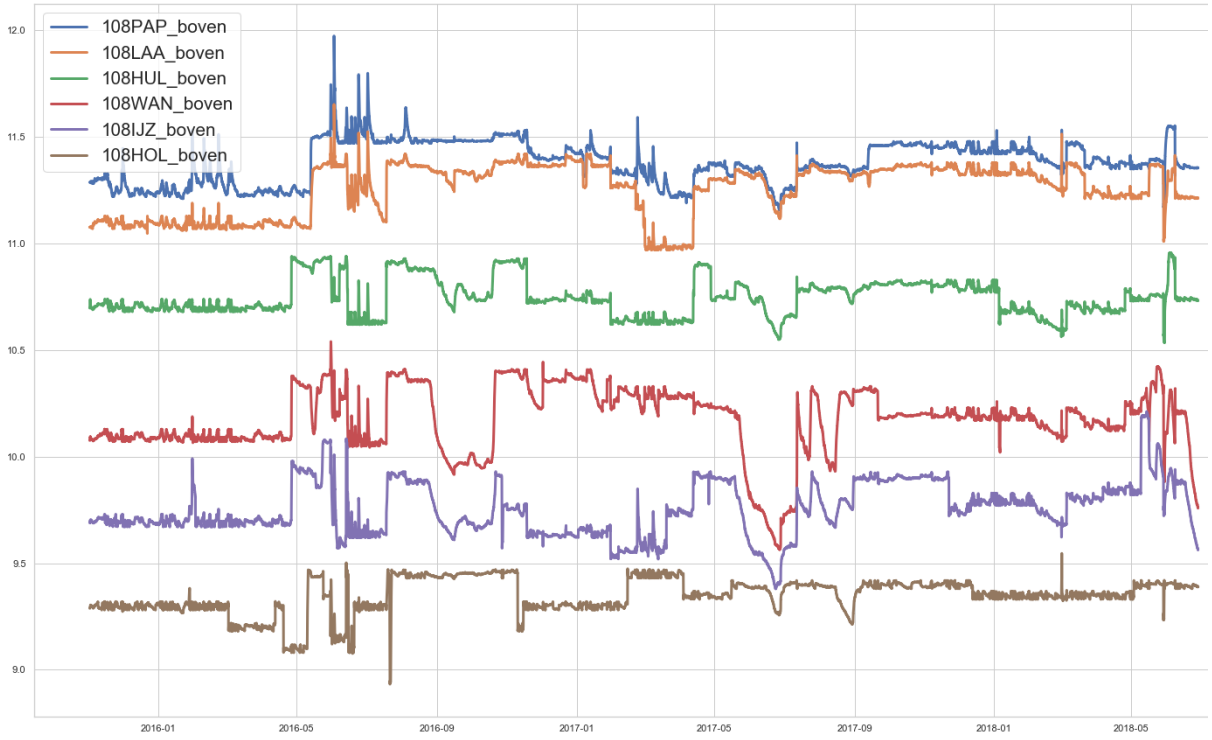


Figure 5.2: Example of a benchmark data set.

The gap creation process was repeated 10 times. We did this, since gaps with different characteristics (like a gap in a relatively constant part of a time series, or a gap which includes a previously present peak or period of high variability) can often be filled in with different rates of success. One example of such a graph with artificial gaps in the data is presented in Figure 5.3.

We used 11 different algorithms to fill up these gaps. A short explanation is given below:

- **Mean**: Univariate algorithm which predicts the mean value for all missing values.
- **Median**: Univariate algorithm which predicts the median value for all missing values.
- **Backfill**: Univariate algorithm which uses the first valid observed value after a period of missing values to fill in the gap.
- **Forwardfill**: Univariate algorithm which propagates the last known value of a time series for all the subsequent missing values.
- **Matrix Factorization**: Multivariate algorithm which factorises the incomplete matrix into low-rank matrices and uses gradient descent to fill in missing values.[1]

The next imputation methods all use the multivariate iterative imputer algorithm[2]. In this multivariate method, missing values are calculated from all other variables which do not have missing values. Features with missing values are modelled as a function of other features. What kind of function should be used can be user-specified. A specific general regression model can be used to this end. This approach is based on the well-known MICE (Multiple Imputation by Chained Equations) approach by Van Buuren and Groothuis-Oudshoorn [46]. A difference between MICE and this idea, is that MICE performs multiple imputations which are returned to the user. Here, only a single imputation is returned.

[1] Uses the implementation of https://github.com/iskandr/fancyimpute
[2] Uses the implementation of https://scikit-learn.org/stable/modules/impute.html

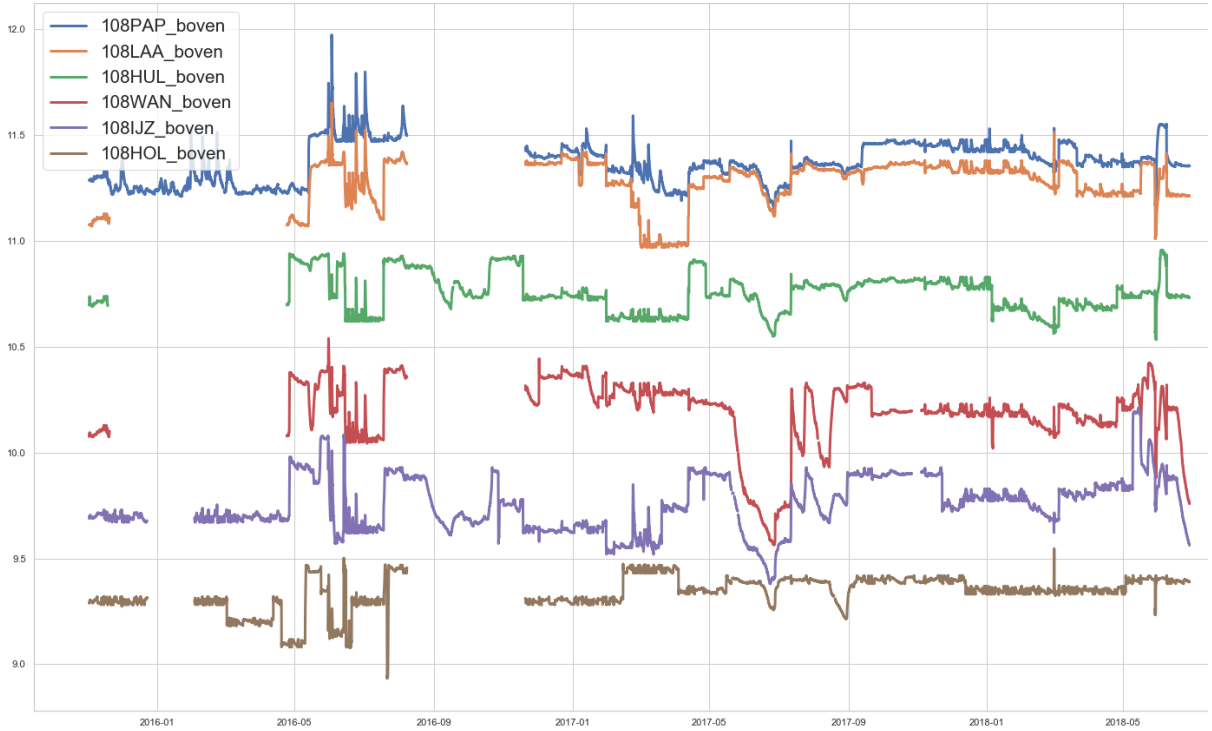


Figure 5.3: Example of gaps we artificially created.

- **Extra trees**: Iterative imputer approach which uses the extra trees model (which are extremely random forests that use random values for the split point) as its estimator.
- **Linear regression**: Iterative imputer approach which uses linear regression as its estimator.
- **Bayesian ridge**: Iterative imputer approach which uses Bayesian ridge (a specific variant of linear ridge regression which also models uncertainty) as its estimator.
- **KNN**: Iterative imputer approach which uses k Nearest Neighbours as its estimator.
- **Random forests**: Iterative imputer approach which uses random forests as its estimator.
- **MTSDI**: Multivariate imputation algorithm in R, especially applicable for time series which are related to each other in a spatial sense.[3]

Two examples of imputed time series based on the gaps of Figure 5.3 can be seen in Figures 5.4 and 5.5.

[3]Uses the implementation of https://cran.r-project.org/web/packages/mtsdi/index.html

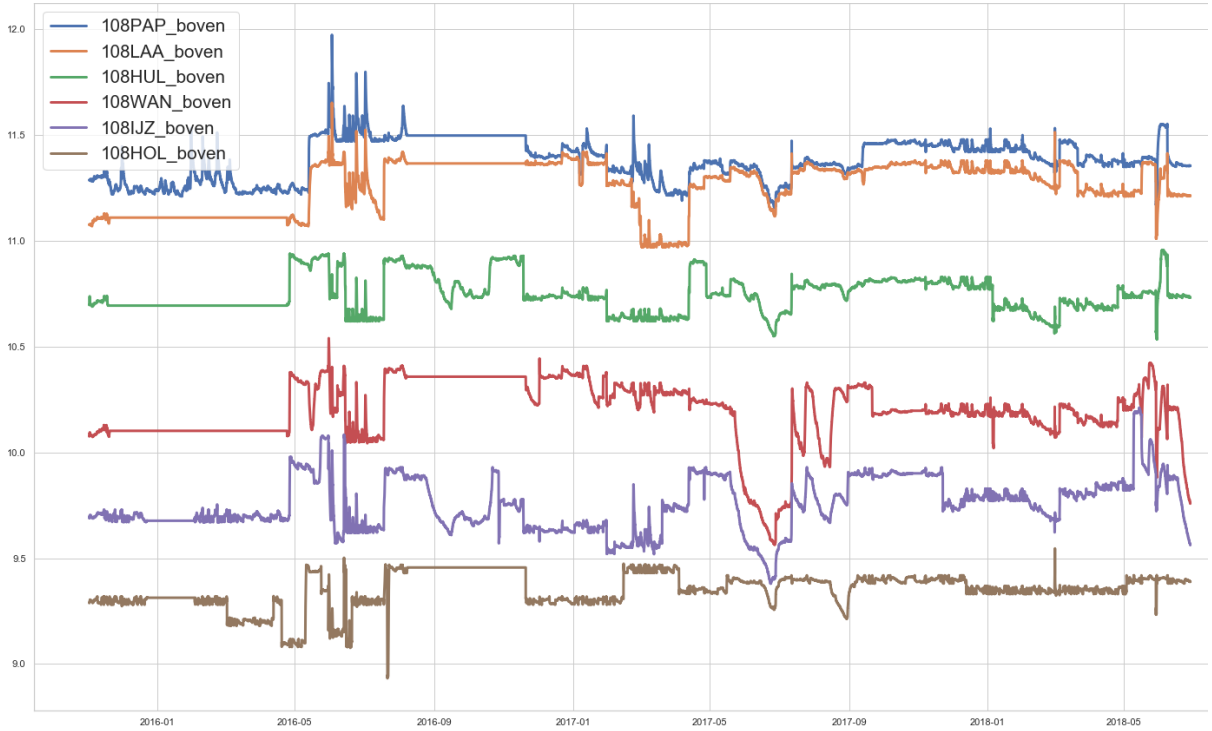


Figure 5.4: Example of the forward fill method applied on a data set with gaps.

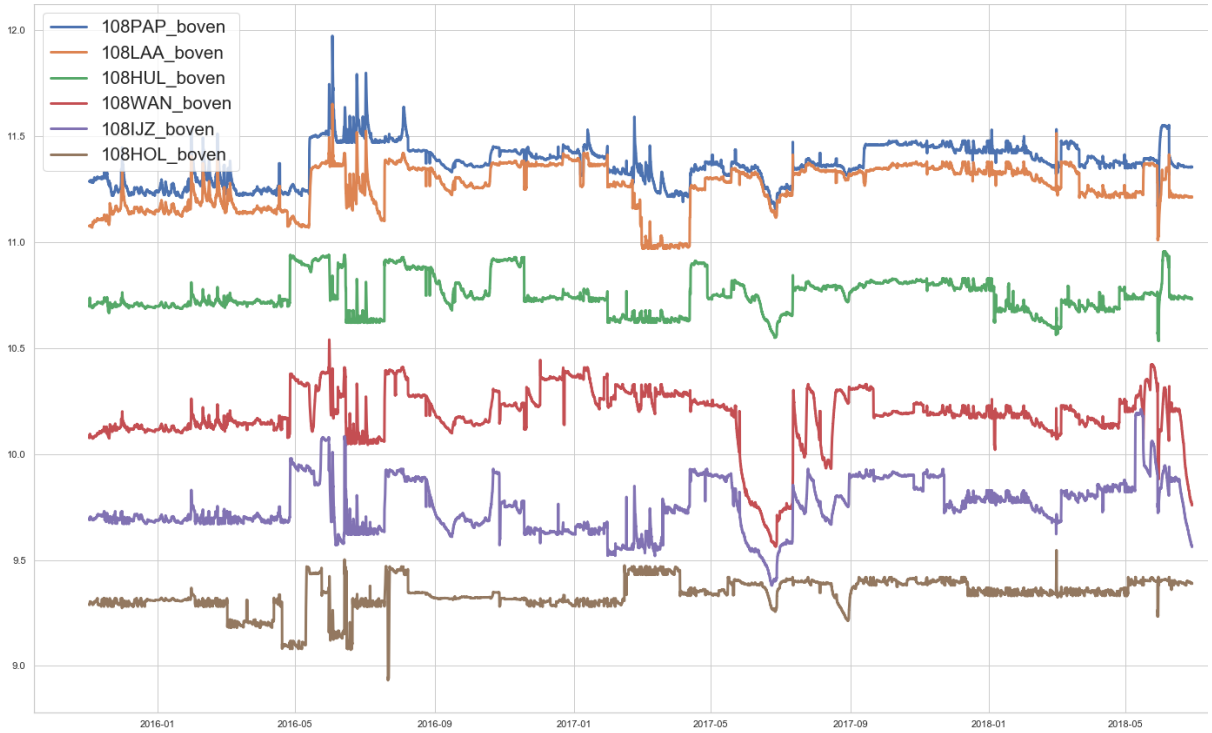


Figure 5.5: Example of the linear regression method in an iterative imputer applied on a data set with gaps.

To quantify the results, we calculated the R^2 scores of the imputated series, given their original series. This score (Equation 5.1), compares the imputation results against a baseline model, which predicts the

mean value of y at every time step. This value normally ranges between 0.0 and 1.0, where a higher value is better. As multiple randomised runs could have different results for the various algorithms, we used bar plots to summarise these results. This procedure was performed on several data sets, a result of one specific set is shown in Figure 5.6.

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{5.1}$$

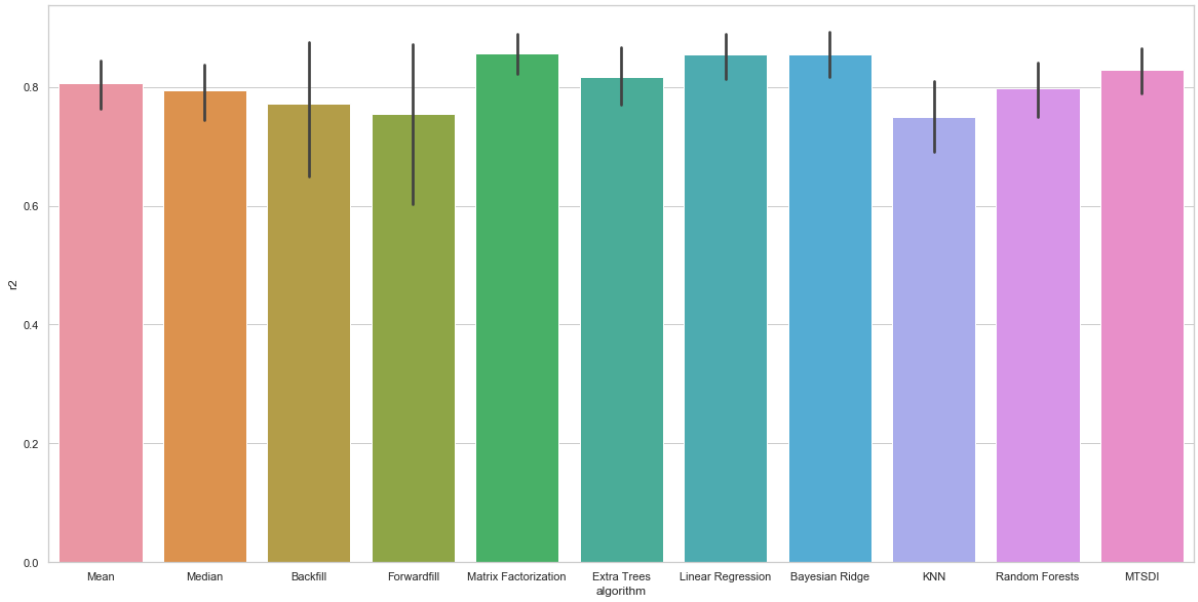


Figure 5.6: Results of the imputation on water height data, using the same sensors as in Figures 5.2 to 5.5. Height of the bar gives the average R^2 for a specific model. The error bars within each bar denote the uncertainty.

5.3.2 Imputation used in the pipeline

As we can derive from Figure 5.6, many algorithms look like they perform decently. Linear regression (in an iterative imputer setting) is one of the best performing algorithms and also has the advantage that it's relatively simple, easily explainable and fast. As a result, we will use linear regression in an iterative imputer for imputation in the pipeline and will use this throughout the whole project.

5.4 Feature engineering

First, we explain the feature engineering step in general and how it is applied to the multivariate modelling experiments. Afterwards we describe how it is used in the univariate experiments.

5.4.1 Feature engineering for multivariate modelling

As of yet, we have only spoken of the values of sensor variables at a specific instant. This is the basis of our approach and is crucial, but can be extended to let models incorporate more information.

One logical extension of raw values often used in time series data is the use of rolling features. These features use a rolling window over past values of an original feature, and calculate a specific function on the values, like the mean or maximum. It is also possible to use rolling lag features. These features use values that were present at previous time steps. This can be useful if there is a relation between two

sensors, but not instantaneously. Since sensors are some distance away from each other, this is likely and thus makes this a good feature candidate.

Experiments were performed to arrive at the ideal features and lag step values. The rolling features we used throughout the experiments were the lag, min, max and mean of lag steps corresponding to 15 and 30 minutes and 1, 2, 4, 8 and 16 hours. Initially it was thought that the models would benefit from having more information to be able to capture underlying relations in a better way. It turned out that the difference between these values and not using history at all was not that big. There still was a difference in most sensors, so giving the models all these features and letting them learn by themselves how they should be used and combined seemed like a sensical approach. By not using too many steps and features, there was a well-balanced trade-off between feature complexity and relevant information encapsulated in the features.

Moreover, historical rain data could be of aid. We used data from Nationale Regenradar and KNMI and incorporated these secondary data sources into the feature engineering step. Then, we tested whether the incorporation of these values was useful. A hypothesis that rain information might help the models could make sense theoretically. Large quantities of rainfall could coincide with a peak in water flow rate, for example. After performing some experiments, we noticed that in water height data this is actually not the case. Even in water flow rate this was not at all prominent. The correlation between the rainfall, even when averaged over a longer time span, and the time series was too low to be able to contribute in the modelling.

5.4.2 Feature engineering for univariate modelling

In the univariate setting, we noticed that most of the features from Section 5.4.1 were not very interesting. An issue of using relatively small window values was that most models tend to behave like persistence models, instead of taking longer history into account. We performed experiments to see if this procedure could be improved. It turned out that the lag, min and max functions were not that useful. On the other hand, mean values over a prolonged period of time were. We experimented with different window sizes. Multiple sensors had slightly different optimal window sizes, and different categories of outliers required different window sizes to be detected in an optimal way. In the end, we selected one feature engineering approach for all univariate models, namely using the mean values of window sizes [256, 512, . . . , 4192].

5.5 Feature scaling

After all the features of Section 5.4 have been created, it is considered wise to scale this data. Instead of using the raw values, standardisation is used to improve algorithm performance. The used standardisation operates on every data column and scales it in such a way that the mean is removed and the data is scaled to unit variance. This kind of feature normalisation can improve the performance of many machine learning algorithms. This step is applied to all the data in the use of every algorithm to enforce consistency between the model runs.

We have to note that the transformations we use to scale the data are fitted to the training data only. So, only the training data is used to calculate the transformation functions. These same transformations are then also applied to the validation and test data. It is vital not to fit these transformations on the validation and test data as well, since this will lead to leaking of feature information, which will lead to an unrealistic setting.

We also tried some other, possibly more powerful transformations. We experimented with Box-Cox [4] and Yeo-Johnson [52] transformations, to make data more Gaussian-like. This could have been beneficial for modelling, but turned out not to be the case. Some initial experiments led us to believe that modelling performance actually worsened by applying these more complex transformations.

5.6 Modelling (ab)normal behaviour

We subdivided the data into a train, validation and test set. All created models were trained on the train set, which consisted of 60% of the available data. The validation set, which was made up of 20% of the data, was used for hyper-parameter selection and/or to prevent overfitting, depending on the used model. The remaining 20% was used as the test set, which we used to compare model performance on.

The models should be able to learn the underlying relations between different variables by training on a large set of data. Ideally, this large set only includes clean data so these relations can be learned as well as possible. That is not always the case, so we have a need for models that are robust with respect to outliers.

The models used can be divided into two subcategories. The first one is regression-based models. In this category, a prediction for a target variable is made. We can compare this against the actual value and then calculate residuals, which are the differences between the predictions and the actual values. It is paramount here that a model should be able to capture the behaviour of a target variable really well. If this is not the case, outlier classification will deteriorate. More specifically, it might be of interest to look exactly how well models should perform to be able to perform decent outlier detection. We could look at some statistical model evaluation measures, which can describe its modelling power. For example, it might be interesting to see what kind of minimum R^2 values we need to have to perform well in the outlier detection procedure.

The other category is more like a direct classification. The direct classification approach looks at data and then directly classifies whether it is considered as an outlier or not. It is now not so easy to calculate something like R^2 to examine model performance.

5.7 Predicting behaviour

When the model has been trained, the test set will be used to make predictions. The different kind of models may predict behaviour in a different way. All the regression-based models (including the neural network-based approaches) work by calculating an expected value for a sensor based on the input variables (possibly accompanied by quantile values). This is different from other models such as isolation forests, which determine directly whether a value should be classified as an outlier.

5.8 Outlier classification

The outlier classification procedure is based on thresholds or quantiles, which is again depending on the specific modelling technique used (and more specifically under which category of Section 5.6 the model can be placed).

To perform outlier detection when using quantile regression, we used the Western Electric rules, made by Western Electric Company [49]. These rules are decision rules which were invented for detecting process instability. We can apply them to outlier detection as well. These rules work by comparing data points against zone limits, which are based on a certain number of standard deviations. We used these same values for the quantiles. We did not apply every Western Electric rule, but we used Rule 1, which classifies each point out of the 3σ zone as an outlier. Moreover, we applied a variation of Rule 2, which states that two out of three consecutive points beyond the 2σ boundary, on the same side of the centre, should be considered very strange.

We were not completely satisfied with the results from Western Electric Rule 2. In many times, this led to a high number of false positives. We tried two other approaches to replace this rule. This was mostly done with the aim to detect drifts, as their gradual change can be hard to detect with Rule 1. The first alternative was to look at consecutive residuals of a series of predictions. When drift is present, we might expect that the residuals are gradually growing in either the positive or negative direction. This

was not of much use to us. This approach is too susceptible to minor inaccuracies in modelling, which results in the inability to efficiently detect drift or other outliers. Even when smoothing residuals over longer periods of time, this was not helpful.

The second alternative is one that we actually used. The main concept is to look at predictions averaged over the span of a day, and check whether this exceeds the averaged values of the 2^{nd} quantile. This idea is described in Algorithm 1. In this manner, minor short-lived errors get smoothed out, but we get a more accurate way of describing a gradual change. A minor disadvantage of this idea, is that there might be some false positives near the beginning and end of the day on which the average was made. Luckily, this does not hinder us to a great extent. When this method was compared to the Western Electric Rule 2, it was performing slightly better.

Algorithm 1 Drift detection by downsampling.

function DRIFT_DETECTION($y_{in}, q2_{upper}, q2_{lower}$)
    $y_{resampled} \leftarrow$ RESAMPLE_TO_DAY(y_{in}) ▷ Resample input series to daily values.

    $q2_{upper} \leftarrow$ RESAMPLE_TO_DAY($q2_{upper}$)
    $q2_{lower} \leftarrow$ RESAMPLE_TO_DAY($q2_{lower}$) ▷ Similar for the previously calculated quantiles.

    $outliers\_daily \leftarrow (y_{resampled} > q2_{upper}) \cup (y_{resampled} < q2_{lower})$ ▷ Drift classification step.

    $outliers\_predicted \leftarrow$ UPSAMPLE_TO_15MIN($outliers\_daily$)

    return $outliers\_predicted$ ▷ Outliers classified per 15 minutes.

Chapter 6

Modelling techniques

In this chapter, we describe the different modelling techniques that will be evaluated in the experiments. The aim of this chapter is to describe the way in which we use these models well enough, so that the experiments are replicable. A general introduction to the modelling approaches will be given as well. This chapter will describe what (hyper-)parameters will be used for the models. Also, some models might be discarded if initial experiments show that they are not suitable for our problem. Afterwards, Chapter 7 will build upon these descriptions, these initial experiments and (hyper-)parameter tuning and use these models on actual data sets to assess their performance.

There are 4 sensors that we will put emphasis on in this chapter. These are the sensors that will be used during the synthetic evaluation of Section 7.2 and are sensors 102BFS, 103HOE, 104OYE and 201D. Of these sensors, we will model the water height on the upper part of the weir. By looking at the test sets of the 4 data sets, which have been validated by Aa en Maas, we could determine that no outliers were present in the test data. This is beneficial for the synthetic evaluation, as already present outliers might interfere with our own created ones.

6.1 General modelling approaches

To convert our extensive literature survey of Chapter 3 into a more succinct collection of used modelling techniques, we will distinguish certain models that we will put focus on. The quantile regression method with a tilted loss function is important to experiment with. More general (V)AR models will be used to see if this can give us some sort of a baseline. These autoregression-based models can be gradually extended to models with increasing complexity, which will result in (quantile) linear regression and random forest approaches. Eventually, other neural networks-based models such as RNNs will be used. Afterwards, the isolation forest method will be used to see if it works well in the data.

We can use the history of a sensor itself to detect outliers in its sensor values. If we use a multivariate approach, we are also able to ignore its own values and base the detection on other sensor time series. An advantage of the first approach is that it is always applicable as no other time series are needed and that large sudden changes in values might be easy to track. An advantage of the multivariate approach is that we can detect changes which happen in one sensor only. Slow, gradual changes over time might be detected well in this manner. For this to function well, we probably need correlated time series.

We discern univariate and multivariate outliers. Univariate outliers concern one specific sensor and could be caused by a sensor defect. Multivariate outliers can be more systematic since more sensors are behaving weirdly. For example, large rainfall could cause this, as this can have effect on multiple sensors and locations.

6.2 Regression-based models with gradually increasing model complexity

Before starting this project, the preferred method for outlier detection at Ynformed was quantile regression by using neural networks. To determine if this method is sensible and is not over-complicating things, we started building models with low model complexity and gradually created more sophisticated ones. Eventually, we will also arrive at such kind of quantile regression models and afterwards we will be able to compare the characteristics of these models.

6.2.1 Autoregressive models

As explained in Section 3.2.1, autoregressive models are often used in practice, but in most cases for regular time-series modelling and possibly prediction instead of outlier detection (although they are used by Leigh et al. [27], for example). It was unsure whether the data and the environment would be suitable for these kinds of models, so experiments needed to be done.

We implemented autoregressive (AR) models as well as vector AR (VAR) models, to compare univariate and multivariate modelling. Intuitively, one might think the multivariate model will be better and more sophisticated, since relations with other sensor data could be modelled, as well as auto-correlation of the sensor itself.

For these kind of models, we needed to use the values of the sensor that we are predicting as well. (V)AR models require the history of these values to calculate next ones. This is different from most other approaches in this chapter, where it is possible to omit the variable that we want to predict from the input variables. Whether this might pose a problem here needs to be seen.

When experimenting with AR and VAR models, we noticed their results actually resembled each other to a large extent. Big sudden jumps could be detected as outliers; other values could not. It turned out that the VAR and AR model were almost doing the same thing. The VAR model used almost the same coefficient values for its own variable that the AR model learned, as is shown in Table 6.1. For variables which the VAR model is not predicting, really small coefficient values were learned. Thus, these other variables had almost no influence. Since the data is sampled at a relatively small interval, data is not changing a lot in successive measurements. Therefore, the models learn coefficients which favour predicting a similar value as currently present. The result of this is that (large) deviations only occur when there is a sudden jump in values. We have more or less emulated a (smart) persistence benchmark at this stage.

Feature	VAR coefficient	AR coefficient
Intercept	0.000	0.000
t-1	1.662	1.666
t-2	-0.658	-0.662
t-3	-0.056	-0.055
t-4	0.055	0.062
t-5	-0.005	-0.011

Table 6.1: Coefficients calculated by VAR and AR.

It was interesting to see that Leigh et al. [27] used similar models and claimed to achieve decent performance. Since we were not really satisfied with the models, their approach was given a closer look. After an inspection of Figure 6.1, it turned out that their performance actually was not that great after all. As can be seen, their multivariate RegARIMA model had similar issues as we had. Just like in our models, we see that many outliers are claimed to occur at data points where the relative difference to neighbouring points is high. This gives many false positives here, but also misses many more subtle outliers (like in the conductivity time series around July/August 2017).

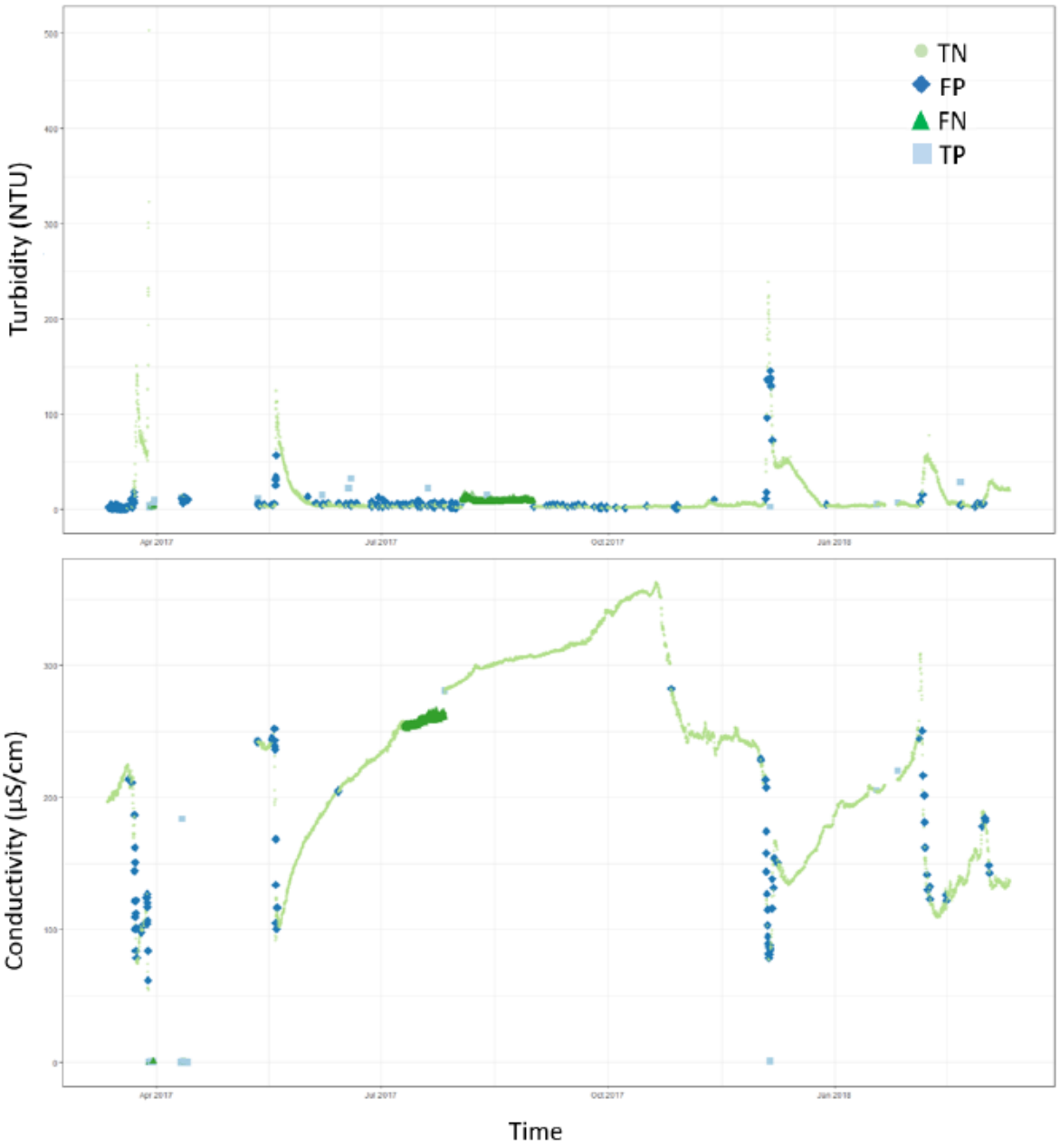


Figure 6.1: Excerpt of figure S5 by Leigh et al. [27]. A RegARIMA model (a model in multivariate ARIMA fashion) is used here to predict outliers, by using a certain threshold on the residuals.

In the end, we conclude that the VAR model will not be used in the multivariate experiments. The AR approach will still be used in the univariate experiments. In the univariate outlier detection, we used the AR model to predict the next value, based on its previous values inside of a certain window. As we can see in Table 6.1, most time steps outside of the first two have little influence. We experimented with larger window sizes than the value of 5 that is shown here, but this did not change much. So, we used a window size of 5 in the experiments. If a predicted value deviated too much from the observed value, it was classified as an outlier. Threshold selection is another important aspect. After we performed some initial experiments, we ended up with a threshold of 0.04. Lower values gave us too many outliers,

higher values gave us too little predictive power. This value is relatively low; Section 7.4.4 will show the implications of this.

6.2.2 Linear regression

A model slightly more complex than a VAR model is linear regression. A slightly higher complexity comes from the fact that we can now use the feature engineering step of Section 5.4 as it is no longer mandatory to use the plain time series as-is. A first assumption about this model, is that it might be too simple to capture the relations in the data well. It could be the case that the correlation between different sensors is unlikely to modelled well by linear regression.

An issue of interest when using linear regression for such kind of an outlier detection problem, is in defining the threshold that is necessary for determining what an outlier is. An approach may be to use the mean and standard deviation of the training data of the sensor of interest to base quantiles on. A problem of this idea, however, is that this will lead to a fixed quantile width for the whole model. This might cause issues, since varying quantile width seems desirable, as uncertainty about the prediction can differ throughout the data.

Instead of just using plain linear regression, we used the Lasso model introduced by Tibshirani [45]. In this algorithm, an α-value can be used to punish overfitting models. This algorithm uses a l_1-error instead of an l_2-error, which entails that coefficient size is punished linearly instead of quadratically. The Lasso function to be minimised is shown in Equation 6.1, where coefficients are denoted as β.

$$\sum_{i=1}^{N} \left(y_i - \sum_j \beta_j x_{ij} \right)^2 + \lambda \sum_j \beta_j \tag{6.1}$$

A main advantage of this approach is that coefficients can actually be zeroed out, which a l_2-approach like ridge regression is unable to do. Since the feature engineering step of Section 5.4 provides us with 104 features in total, this is useful. For example, when applying this on sensor 104OYE, only 11 of the 104 calculated coefficients were non-zero. One of the 4 underlying time series was not used at all, all mean values were discarded and only one lag value was kept.

We used the validation set on the 4 used sensors to determine the ideal value for α. Figure 6.2 describes the results of this tuning procedure. Validation loss for α-values in [3.0, 1.0, 0.3, 0.1, ..., 0.001, 0.0003] were reported. To display this, a logarithmic scale seems wise. It is useful to use an α-value which scores well, but also is relatively large. This is the case, as it will punish overfitting, while still keeping decent performance. In Figure 6.2, we notice that different sensors have different α-values corresponding to a minimisation of the validation loss. Still, we do see that they are all in proximity to each other. Based on this graph and on the actual modelling ability of the resulting values (which indicated that a straight line could perform okay in some cases, but is certainly not desired behaviour), we conclude by using an α-value of 0.03 throughout the experiments.

We could have used the linear regression in a non-quantile way. Then, for each data point, we need to calculate the residual and use a threshold to decide whether it will be classified as an outlier. Instead, we opted for a quantile approach so we could use the same classification rules as in Section 5.8. To calculate the quantiles, we first calculated the standard deviation of the to be modelled time series. Then, we added or subtracted this quantity times a scalar value to the mean prediction to create the quantiles. This procedure to calculate the different quantiles q_i is shown in Equation 6.2. This ad-hoc like approach somewhat mimics the quantile results that the algorithms of Section 6.3 are able to achieve.

$$q_i = \hat{y} \pm \frac{i}{2} * \sigma(y_{train}) \quad | \quad i \in \{1, 2, 3\} \tag{6.2}$$

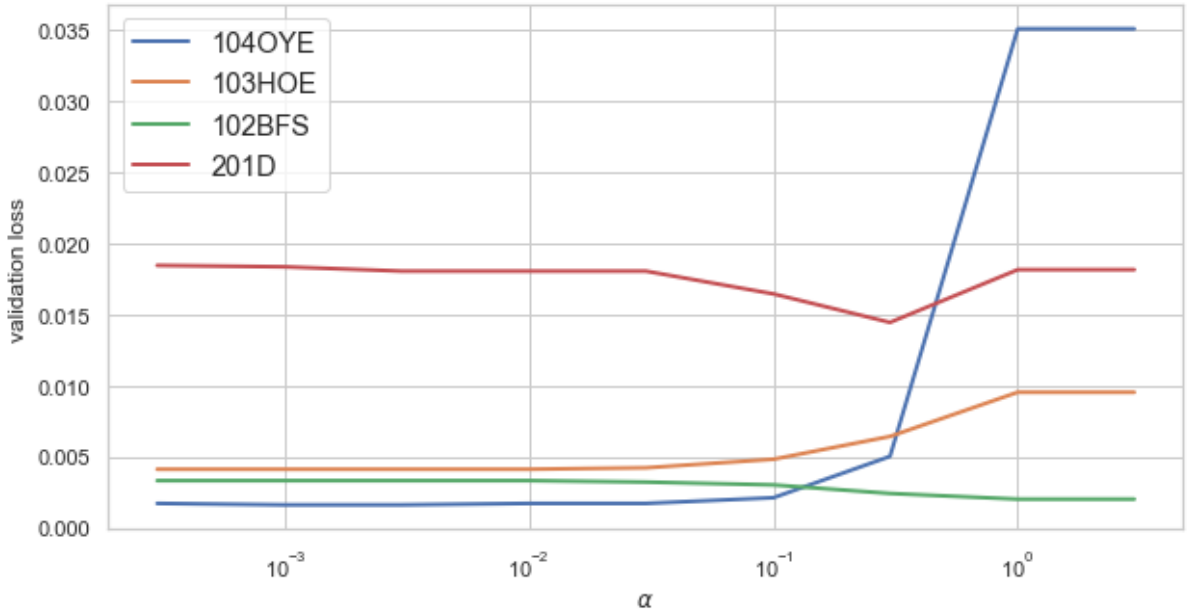


Figure 6.2: Validation loss (mean squared error) for different α-values tested on the 4 sensors in linear regression.

6.2.3 Quantile regression forests

The quantile regression forests algorithm from the scikit-garden package[1] was used, with some additions to make it more like the code from the package in R, created by the inventor of the algorithm Meinshausen[2]. This was done, as it greatly improves prediction times, which is almost a necessity for the broad experiment setup.

Initial experiments indicated the need for a long run-time to perform outlier detection, both the case in the Python and in the R implementation. It seems probable that these speed issues are due to the algorithmic complexity of quantile regression forests. This approach could still be applicable to our project, but we need to keep in mind that scaling to large data sets (in number of training examples as well as in number of predictor variables) could be infeasible, especially if models need to be retrained often.

Some parameter values of the algorithm were experimented with. It was not possible to do this in an exhaustive way, as the program is relatively slow (in training as well as in prediction, even with the previously mentioned speed improvements). To achieve a large enough variability and decent enough performance, multiple trees need to be combined. We used 1000 different trees in total. For each tree, we used the same parameter settings: A node needs to have at least 40 samples in it for it to be considered for a split, a resulting leaf node must have at least 20 samples and the maximum number of considered features per split is one third of the total number of features. All of these steps are done to ensure a plausible and realistic running time, while still keeping the model performant.

6.3 Neural network-based approaches

In the simpler models of Section 6.2, we encountered some practical issues. In real-life scenarios and use cases, the standard quantile regression approach for outlier detection often used by data scientists from Ynformed is neural network-based. To alleviate the problems of the simpler models, it seems like

[1] https://scikit-garden.github.io/examples/QuantileRegressionForests/
[2] https://cran.r-project.org/web/packages/quantregForest/quantregForest.pdf

this more complex idea might be wise to apply to our case as well. After applying a default method for neural networks, it is also interesting to see if we can improve on this with additional, possibly more complex models.

When creating a neural network, it is not at all trivial to decide what hyper-parameter values to use. In the case of neural networks, the tuning of the architecture of the network is a fine craft. Exploring all different possibilities is impossible, as the possible size of such a network is limitless. However, for our modelling tasks, it is probably not needed to create very complex networks, since we are not dealing with huge amounts of data. This can be the case in image or video classification, for example. In practice, it is often the case that some architectural varieties are tried out in a non-structured, trial-and-error, fashion. This is sometimes comically referred to as "grad student descent" [13].

In an effort to do neural network architecture tuning in a more advanced and systematic way, we used the Hyperband algorithm, which was introduced by Li et al. [28]. This algorithm is more sophisticated than a random search process over the possible hyper-parameters. The main idea is that many different hyper-parameter combinations are tried out to see which works the best. Just like in a random search process, these combinations are selected randomly, initially. A great contrast, however, is that these models are initially only trained in just a few epochs. After many models have been trained, the best ones are selected and will be trained for further epochs. This process is continued until only one model remains.

Then, a next round is started. Just like in the previous round, some random configurations are tried out, but now they are trained for more epochs. Also, fewer configurations are tried then in the previous round. Again, this will eventually lead to one winning model. Some further rounds may ensue, depending on some model settings. In the end, many different model architectures have been evaluated, where some will have had more epochs to be fitted than others.

We repeat this whole process three times, to be sure that many different network architectures are evaluated. Also, for each configuration, five models with different initial weights are trained and their individual performance scores are averaged. This is done, as random initial configurations do have a non-negligible effect on the model performance. As a result, we have a thorough hyper-parameter selection procedure, which can improve on a standard random search approach.

In each of the following neural network sections, we will elaborate further what specific architecture parameter settings will be explored by the Hyperband algorithm, as this is not the same for all models. Results of this tuning step will also be reported, so Chapter 7 can focus on the results retrieved by using a high-performing hyper-parameter configuration.

In the multivariate experiments of Chapter 7 we averaged the predictions of 10 different neural networks. This ensemble approach is done as random weight initialisation has noticable influence over the created model. In the univariate experiments this number is lowered to 5, to still keep running times plausible as many more predictions need to be done. These extra predictions are needed because the input of the testing data changes per outlier time series, which was not the case in the multivariate modelling.

6.3.1 Quantile regression: Multi layer perceptron with multiple outputs

For the first quantile regression model using neural networks, we will focus on using a multi layer perceptron (MLP). This is a neural network with possibly multiple hidden layers, that only uses dense neural network layers.

In applying the Hyperband algorithm, we used 5 executions per trial. This means that per setting, 5 models are trained and their results are averaged. The models were trained on the train set and were scored on the validation set. The whole Hyperband algorithm was run 3 times, with the maximum number of epochs set to 30 and the scaling factor to 3. This resulted in initial models being trained for 2 iterations. This is a small number, but due to the Hyperband approach, models could also be trained for longer durations in subsequent rounds or when the selection in a current round is narrowed down.

In the end, 270 trials were run. Some of these trials have the same hyper-parameters. This could occur by chance in multiple runs, but a more common cause is that well-performing configurations are trained further. We looked at the best runs and filtered out runs with the same parameters, so we only looked at the best one of them.

In this MLP setting, we let the Hyperband algorithm select the number of layers (1, 2, 4 or 8), number of units per layer (which could differ per layer, 16, 32, 64, 128 or 256), dropout (0.0, 0.1, 0.2, 0.3 or 0.4) and learning rate of the network (0.005, 0.001, 0.0005, 0.0001, 0.00005 or 0.00001). As one can easily calculate, trying all different combinations of these parameters is infeasible.

In the algorithm runs, we used early stopping with a patience value of 5. This means that model training is stopped, after the performance (scored on the validation data) has not increased after 5 full iterations. The training loss will still decrease, but the validation loss might not. This approach of early stopping combats overfitting. Moreover, we used a batch size of 128. This means that we are not using default gradient descent in the neural network, but are using mini-batch training, as explained by Masters and Luschi [34].

We assumed that it was possible to run this for every sensor, and get some kind of a general network architecture that works for each sensor. In theory, this could work, as the amount of data being fed to the network is equal in all the cases. As a way of ranking, we ordered the models, descending, by validation loss. This is the pinball loss where all the quantiles are taken into account, as explained in Rodrigues and Pereira [41]. The results for the four different sensors are shown in Table C.1.

Whereas we initially thought we could find one general network architecture that works in all cases, this is clearly not the case. There seems to be some correlation between the validation loss and the network complexity. For example, sensor 104OYE can be modelled relatively well. All of the best networks only use one layer. On the other end of the spectrum we see 102BFS (a sensor which was selected especially because of the low correlation with other sensors, so we could see if this (severely) impacts results), which has high validation loss and needs more complex models.

On account of these results, we decided to use a different architecture for the different sensors. The best dropout, learning rate and number of layers was picked. One value for the number of neurons was used, to be able to generalise a bit more. The concluding architectures are shown in Table 6.2.

Sensor	Dropout	Learning rate	Number of layers	Neurons per layer	Average validation loss of final model
104OYE	0.4	0.0005	1	128	0.1820
103HOE	0.0	0.005	1	256	0.8790
201D	0.4	0.005	2	128	0.9580
102BFS	0.4	0.00005	8	128	1.5330

Table 6.2: Final MLP model architectures per sensor

6.3.2 Quantile regression: Perceptron model

A baseline neural network model in the form of a perceptron model was created. We created a neural network without any hidden layers. There are only direct connections between the input nodes and the output nodes. Since the hidden layers disappear, we also do not have a possibility for applying dropout and differing the neurons per layer anymore. Thus, the only thing that can be tuned is the learning rate. Again, we use a batch size of 128. In the tuning process, we did not use the Hyperband algorithm. Since we are only tuning 6 different learning rates, an exhaustive search is now possible.

Sensor \ Learning rate	0.005	0.001	0.0005	0.0001	0.00005	0.00001
104OYE	**0.2194**	0.2220	0.2247	0.2248	0.2273	0.2261
103HOE	**1.1747**	1.2159	1.2969	1.3123	1.4993	1.9265
201D	**1.6964**	1.7734	1.7738	1.8762	1.8783	3.0741
102BFS	**1.1660**	1.2191	1.2326	1.2569	1.2695	1.3049

Table 6.3: Learning rate parameter tuning for the perceptron model. Average validation loss of 10 runs. Best value marked in bold.

As we can see in Table 6.3, is seems that a large learning rate works best for this kind of model. Consequently, it is a logical choice to just use a learning rate of 0.005 for all the perceptron models. Moreover, we see that the validation loss is (much) higher than in the models of Table 6.2, with a notable exception of sensor 102BFS.

6.3.3 Quantile regression: RNNs

In this section, LSTMs and RNNs will be experimented with. Instead of using normal dense layers in the network, we now used RNN layers. Again, a tuner approach was used. We let the tuner decide if a GRU or LSTM kind of RNN layer should be used. A difference between the tuning of Section 6.3.1 is that we now use at most 4 layers, because of the large run times needed when tuning these models. Also, we use a batch size of 2048 instead of 128. This speeds up run times dramatically, which was needed to run these experiments in a doable time. This will surely have an impact on the ideal learning rate, but since we are still tuning this in a broad range, no problems regarding this should arise. The last big difference is that we now only use a window size of 32, instead of a maximum window of 64 as was explained in Section 5.4. So, we now use the previous 8 hours of data, instead of the previous 16 hours. This was also done for run time improvements and did not seem to impact the model quality. A main difference, however, is that since we have a RNN, all these 32 values are used in every step. This is the case, as RNNs need time series in their original form. Also, this disallows us from explicitly modelling features like the minimum and mean features. The model should be able to learn these relations from the data. The results of the evaluation runs are shown in Table C.2.

Again, it seems to be the case that there is no general, best overall network architecture. Moreover, it seems that neither LSTM-layers nor RNN-layers work best for every network. Like in Section 6.3.1, we will use a different network architecture per sensor. Similarly, for each different sensor, we picked one value for the number of neurons, to generalise a bit more. The final used models are shown in Table 6.4.

Sensor	RNN type	Dropout	Learning rate	Number of layers	Neurons per layer	Average validation loss of final model
104OYE	GRU	0.1	0.0005	1	256	0.2060
103HOE	LSTM	0.0	0.005	1	256	0.9401
201D	LSTM	0.4	0.001	4	64	1.3292
102BFS	LSTM	0.2	0.0001	4	64	1.2330

Table 6.4: Final RNN model architectures per sensor

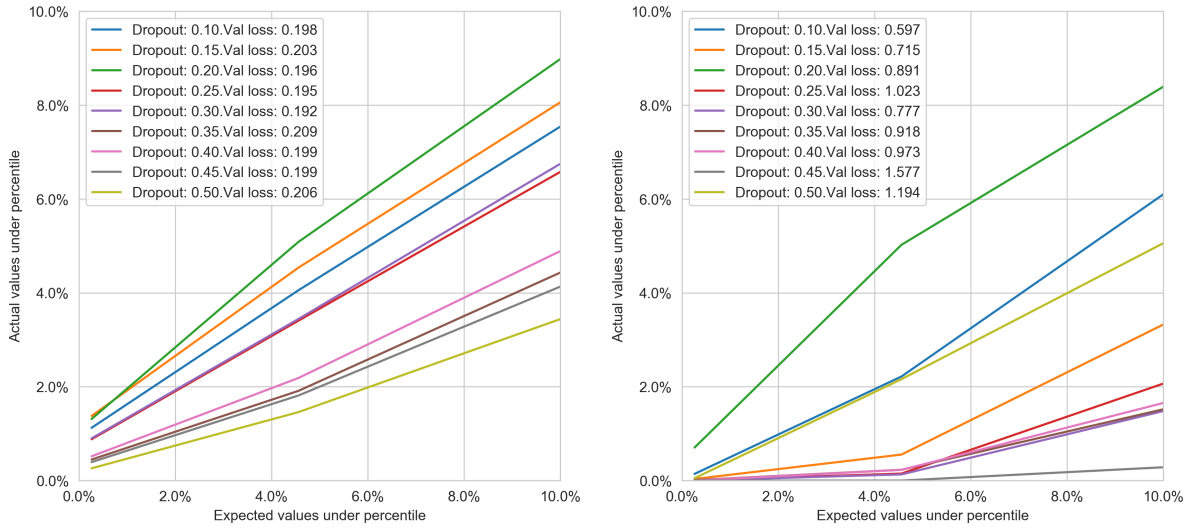
6.3.4 Discarded model: Quantile regression: Dropout during prediction

In Section 3.2.3 and Section 3.3, a method to derive quantiles by using the neural network technique dropout is described. This method is similar to the previously described models of Section 6.3 in the sense that neural networks are used, but dissimilar in the sense that now the tilted loss function is not used anymore and quantiles can be calculated by making predictions multiple times.

Two different methods to implement this approach have been tried. The first used all the different predictions (we used 1000) and creates quantiles based on picking a sample corresponding to a specific quantile of the ordering of these predictions. The second method uses the standard deviation of all these predictions and adds them multiplied by a multiplier to the average predicted value to calculate the quantiles. For different quantiles, a different multiplier (1, 2 or 3) was used. We have used the second method, as the first method was relatively noisy. This could have been circumvented by using many more than 1000 predictions, but a large downside of this is that experiments would have been very slow.

There are two main problems in this approach. Aside from the model being relatively slow, there is a more fundamental issue. In this model, we are hugely relying on the dropout to create an uncertainty interval. To do this accurately, the choice of the dropout parameter is paramount. The higher the dropout parameter, the more wide the distribution of calculated values can get. Thus, to create reasonable intervals, the dropout parameter needs to be set to a right value. A major issue of this, is that the dropout parameter might also be very important for creating an accurate model. As we have seen in the previous sections, a specific dropout value might be required for modelling a certain sensor.

Some experiments to tune this dropout value have been performed. We trained a neural network on outlier-free testing data of a sensor set which is known to behave normally during the span of this data. Then, prediction is done on this same test data, with dropout used during prediction. The idea behind this, is that we may now model the data distribution by varying the dropout parameter. This was done on two data sets and the results can be seen in Figure 6.3.



(a) 104OYE

(b) 201D

Figure 6.3: Dropout tuner results. Average over 3 runs with 1000 predictions each.

In this figure, we experimented with 9 dropout values. For each value, we trained 3 neural networks and calculated 1000 predictions using each network. Based on the quantiles corresponding to all these predictions, we took note how many values fell outside of the quantiles corresponding to the 1^{st}, 2^{nd} and 3^{rd} standard deviation (both on the upper and lower boundaries). This yields us percentages observed at the expected percentages 0.26%, 4.56% and 31.74%[3] for respectively the 3^{rd}, 2^{nd} and 1^{st} standard deviation. The graph is constrained between 0% and 10% to be able to depict this clearly. In an ideal scenario, we would have a straight line, as that would indicate neither overestimation nor underestimation of the underlying distribution. In both images, there is no straight line. The green line corresponding to a dropout value of 0.20 seems to fit the best on average, but in Figure 6.3b, this model

[3]These values are the sum of expected percentages beneath and above respectively the lower and upper boundary, which correspond to (0.13, 99.87) (2.28, 97.72) and (15.87, 84.13) for the 3^{rd}, 2^{nd} and 1^{st} quantile.

has a relatively large average loss.[4]. Also, it is expected that lines corresponding to low dropout values are leaning more to the upper left, whereas high dropout values are expected to lean to the bottom right. This is expected, due to higher dropout values leading to more variation in the predictions. We can see that this ordering from upper left to bottom right is not strictly ascending as expected. This could be problematic, although the big variation in created networks makes it hard to make any solid claims about this issue.

We see that making a correct decision which dropout value to use is not really doable. Thus, we concluded that this method will not be used in the further experiments.

6.3.5 Discarded model: Autoencoder

In Section 3.2.5, we described a general approach of using autoencoders for outlier detection. This general approach may be a bit dissimilar from what we aim to solve. Like in a manufacturing process or in a sewage pumping station, our data may be high-dimensional because time series of multiple sensors might be used in an outlier classification. Contrary to this manufacturing process, however, these sensors are all different mechanical entities which do not belong to the same system. When we input multivariate data like this in an autoencoder, it will try to reconstruct the sensor values for all of these sensors, instead of just focusing on one sensor which might be of interest to us. We have to note that we cannot retrieve the root cause of this outlier. Domain experts who would use a system like this in practice still need to perform a lot of manual analysis to determine and resolve the actual underlying issue. The fact that autoencoders for outlier detection should be trained on outlier-free data might also prove troublesome.

It is also possible to calculate the reconstruction error of a single series. This can be done both in the multivariate and in the univariate sense, if we only look at the reconstruction error of a sensor value and discard the reconstruction of the derived features. This is not really a proven method for outlier detection, but is a novel approach. During the feature engineering step of Section 5.4, we use lag variables. So, the reconstruction phase of the autoencoder will also recreate the time series of these lag variables. In this novel approach, the reconstruction error will only be based on the reconstructed original time series, not taking the reconstruction of the lagged time series into account. Whether this holds theoretically and mathematically is unsure, which makes it a questionable method.

Another downside of using multivariate autoencoder modelling is that it is mandatory to incorporate the own history of a sensor, as the reconstruction error can not be measured if the own history is removed. This kind of modelling has some issues, just as we have seen in the autoregressive methods of Section 6.2.1.

Another option is to perform outlier detection in an univariate setting only. Then, a specific target sensor is considered as the whole system. Again, we can reconstruct the whole system of lagged time series. Now, it is possible and sensible to calculate the reconstruction error of this entire system. This approach is more alike to the usual practice of performing outlier detection by using autoencoders, since we returned to the whole system setting. On account of this, we are now more sure to have an approach with higher validity.

Although we first deemed the autoencoder approach to be promising, we must note that some of these approaches are dubious. The multivariate approach with focus on one time series as well as the approach based on the reconstruction error of the whole system is questionable both theoretically and practically. In the end, it is probably not advisable to use autoencoders for multivariate outlier detection in our project. We can still use the univariate idea, however. But, since some initial experiments showed relatively poor performance, we discarded this method.

[4]These losses cannot be directly compared with tilted losses from before, as we are now using the mean squared error for each model.

6.4 Miscellaneous models

Some techniques dissimilar to the previously discussed regression-based and neural network-based approaches were also used. They are discussed in the following sections.

6.4.1 Isolation Forests

At first, we thought the isolation forest model to be promising. After some initial experiments, we encountered some issues. When considering multivariate feature sets, we will encounter the same problems as when using autoencoders and VAR-models. This means that we can only look at outliers of a whole system. It is not possible to only predict outliers for a specific sensor, since we will only get an outlier label for the total system. This caused us to only know if there are outliers in the combined feature set, which is problematic when trying to determine which sensor is causing the outlier. Also, it is again mandatory to incorporate the own history of a sensor.

So, like in the autoregression Section of 6.2.1 and the autoencoder Section of 6.3.5 we conclude that we should not use this approach in a multivariate setting and should only apply it to the univariate one.

We have done some hyper-parameter tuning to determine the ideal value of the contamination parameter. This value indicates the proportion of outliers in the data set and is used to determine the threshold during the learning phase. If we set the contamination value too low, we will detect few outliers. If it is set too high, the precision of our model will drop. Our experiments suggested a value of 0.07, which seems like a decent trade-off between precision and recall. These experiments have been performed using jump and drift data sets; extreme value outlier detection performed poorly regardless of the parameter choice.

6.4.2 Discarded model: Distance-based methods: K-Nearest neighbour approaches

As we have seen in Section 3.2.6, there are numerous distance-based methods for outlier detection. Performing experiments on all these mentioned methods is too extensive and time-consuming for this project, which is why we consider it out of scope. Instead, we decided to focus on the most promising approaches, which include the KNN-based approaches. These methods were also covered by Leigh et al. [27] and Talagala et al. [43]. They note that the KNN-methods are preferred if outlier clusters are present in feature-space. An example of this is the recurrent spiking of a time series by the same value. This is exactly something which can occur in our data, as there could be technical issues with sensor machinery, for example. For that reason, we implemented both the KNN-SUM and KNN-AGG algorithms. Some minor experiments with KNN-based regression were also performed, but since obtained results were clearly inferior compared to the neural network-based methods, we decided not to pursue that avenue. Similar to Section 6.2.1, 6.3.5 and 6.4.1, we have the issue of only examining outliers of a total multivariate system.

When we experimented with these implementations, we noticed that the initial results were not very promising. Therefore, we discarded this method.

Chapter 7

Results and evaluation

In this chapter, we describe the results and evaluation of the models. First, we give a general overview of the metrics that we use. The multivariate modelling results of the synthetic evaluation will follow. We then use statistical tests to compare the model results for statistical significance. Afterwards, we describe the univariate modelling results. Next, we compare the univariate and the multivariate models by testing for significant differences. We conclude the chapter by looking at the practical performance and impact of our models.

7.1 Metrics

Before we discuss the results of the experiments, it is first important to describe the performance metrics which we will use to compare all the model results. These metrics describe model performance based on the number of occurrences of true and false positives and negatives the model has classified. These values can be organised as in Table 7.1. In the context of our research, a true positive is an (added) outlier that is correctly identified. A false positive wrongly indicates that an outlier is present. A false negative is a missed outlier and a true negative is a normal or expected data point which was classified as such.

		Predicted	
		Positive	Negative
Actual	Positive	True positive (TP)	False negative (FN)
	Negative	False positive (FP)	True negative (TN)

Table 7.1: Confusion matrix layout.

Some general and interesting metrics are defined in the following manner:

- **Accuracy** is the proportion of correctly classified cases to the total number of cases.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.1}$$

- **Positive Predictive Value (PPV)**, also known as **precision**, measures the probability that a case predicted to be positive indeed is positive [43].

$$PPV = \frac{TP}{TP + FP} \tag{7.2}$$

- **Negative Predictive Value (NPV)** indicates the probability that cases predicted to be negative are indeed negative.

$$NPV = \frac{TN}{TN + FN} \tag{7.3}$$

- **Sensitivity (Sn)**, also known as **recall**, is the proportion of positive patterns being correctly recognised.

$$Sensitivity = \frac{TP}{TP + FN} \tag{7.4}$$

- **Specificity (Sp)** is the proportion of negative patterns being correctly recognised.

$$Specificity = \frac{TN}{TN + FP} \tag{7.5}$$

- **F_β-score** is a combination of recall and precision, where β indicates how many times recall is considered as important as precision [6].

$$F_\beta = (1 + \beta^2) \times \frac{precision \times recall}{(\beta^2 \times precision) + recall} \tag{7.6}$$

It is probably not useful to look at accuracy, since true negatives will dominate the data set. This is intuitive, since outliers are by definition rare. Metrics like the PPV and NPV might be better indicators, although NPV still has the same issue. On the other hand, when using synthetic evaluation, we may have many outliers.

Which metric we will put the most focus on depends on whether we deem minimising false positives, or minimising false negatives more important. In our setting, minimising false negatives and thus putting more emphasis on correctly recognising positives (outliers) is more important. This is the case, since in an implemented real-life scenario of this system, outlier detection will lead to triggers for database maintainers and data specialists. They need to verify whether outliers are correctly detected. Since this flagging of data will be validated by domain experts, it is probably better to have a false alarm, than to miss an outlier. Thus, a metric like recall might be more important than precision. As we do not want to overload domain experts with false positives, precision is still of some importance and it might be wise to consider the F_β-score, where we can combine these two values in the manner we prefer. A value of 2 for β is fairly common and seems like a valid approach for the experiments. Nevertheless, we will list all these metrics for the test runs.

Aside from these conventional metrics, we will also show some we created ourselves for the synthetic evaluation. They are dependent on the specific kind of outlier superimposed. For outliers which occur over a period of time, such as drifts or jumps, we will record if the whole period of outliers is missed (so, not a single value in the period has been marked as an outlier). On top of that, we will record in how many time steps (or after what drift value) a specific drift is detected.

The fitted models are used on the test set to assess model performance. Evaluation of the results will be done by using simulation.

7.2 Synthetic evaluation: Multivariate results

The synthetic simulation evaluation method entails that we altered the data ourselves to simulate outliers that might happen in reality. Such a method is also used by Perelman et al. [36]. They superimposed (parametrized) simulated events on existing patterns, to handle the issue that certain events were hard to prespecify. This is also an interesting method for us, since it enables evaluating to what extent the used algorithms can handle certain outliers. Also, since our approach is in the unsupervised setting, we

do not have much labelled data to validate or evaluate model performance. By using simulation, we are able to generate exactly those outliers that we want in an easily reproducible and controlled setting, which can be used on multiple data sets and used for multiple algorithms that we wish to compare.

7.2.1 Synthetic outlier classes

We manually introduced multiple kinds of outliers in the data. We perused the outlier definitions of Versteeg and de Graaff [48] and Leigh et al. [27] to get an idea for outlier categories that are used in literature on water time series data.

The synthetic outlier categories we created and used in the synthetic simulation are:

- **Jumps:** A period of data which is increased or decreased by a constant value. After the period has ended, the data values return back to the original range.
- **Extreme values:** Isolated data points which are increased or decreased by a constant value. This category is alike to jumps, but is fundamentally different, in the sense that the window is greatly reduced to only one value.
- **Linear drift:** The occurrence of a series which has a gradual linear trend upwards or downwards. Unlike jumps, drift occurs over time and is not instantaneous.

We implemented some other categories that were not used in the experiments. Flatlines, extra liveliness and two techniques for lowered liveliness were created. In discussion with domain experts, we decided which categories are the most important and will thus be the ones that we will focus on. Apparently, unnatural flatlines in actual water height and nonzero flow rate data are not possible, since the data storage system already has checks to prevent this from occurring. Also, the liveliness categories are deemed less important. Thus, we focused on jumps, extreme values and linear drift because these were deemed to be important by the domain experts. These categories will be the ones used for the synthetic model evaluation approach.

At first, we attempted a method where the same outlier type was superimposed multiple times on one time series that we were checking for outliers. Each outlier was similarly parameterised in terms of width and value. Depending on the outlier type, this approach may yield problems. Suppose we are currently interested in modelling drift. To enable meaningful comparisons between models, we need a large enough set of outliers. But, if we want to create many drifts which are not trivially small in the data, we may encounter overlapping drifts, which will result in very unnatural behaviour. The same might be said for jumps. For extreme values, however, this is far less of an issue, because these data points are by definition isolated, with a width of 1. Thus, they will not overlap and this approach is valid for this outlier category.

To perform synthetic evaluation for jumps and linear drift in a smarter way, we created multiple test series with different outliers in it. To be more specific, we used one specific drift or jump and then moved this outlier throughout the data, with each movement yielding us a new series. We alternated between outliers oriented upwards and downwards. In this way, we can create a model and create predictions in the usual way, but we do need to compare each of the created test series against the predictions. Since this is computationally inexpensive, it is not an issue. For each test case, we created 100 of these series. We used multiple outlier generation settings and will show results for multiple outlier values. These values (in meters) were 0.02, 0.05, 0.1, 0.2, 0.3, 0.5 and 1.0. Also, jumps had the duration of approximately 1.5 months, whereas drifts were approximately 6 months wide. Examples of these outliers are given in Figure 7.1 and this idea is described more formally in Algorithm 2.

Algorithm 2 Synthetic outlier generation

function CREATE_OUTLIER_SCENARIO($y_{test}, value, duration, number\_of\_series, type$)
  $num\_timesteps \leftarrow$ COUNT_TIMESTEPS(y_{test})

  if type is extremes **then** ▷ $number\_of\_series$ is interpreted as $num\_outliers$ here.
    $timesteps\_between\_extremes \leftarrow \frac{num\_timesteps}{num\_outliers}$
    $outlier\_times \leftarrow$ RANGE$(0, num_timesteps, timesteps_between_extremes)$
    $extreme\_series \leftarrow$ REPEAT$(value, num_outliers)$ ▷ Create $num\_outliers$ of size $value$.

    for i in LENGTH($number\_of\_series$) **do**
      if IS_UNEVEN(i) **then**
        extreme_series[i] *= -1 ▷ Alternate outlier direction.
    $y\_test[outlier\_times] += extreme\_series$ ▷ Extreme value is added at distinct timesteps.
    return y_test ▷ Only one time series is returned.

  else ▷ Jumps or drift category. First calculate begin and end time per outlier.
    $timesteps\_between\_starts \leftarrow \frac{num\_timesteps - duration}{number\_of\_series}$ ▷ Evenly space starts.
    $begin\_times \leftarrow$ RANGE$(0, num_timesteps - duration, timesteps_between_starts)$
    $end\_times \leftarrow$ RANGE$(duration - 1, num_timesteps, timesteps_between_starts)$

    $list\_of\_series \leftarrow []$ ▷ Initialize list of adjusted series.

    for i in LENGTH($number\_of\_series$) **do**
      if IS_UNEVEN(i) **then**
        $value* = -1$ ▷ Alternate outlier direction.

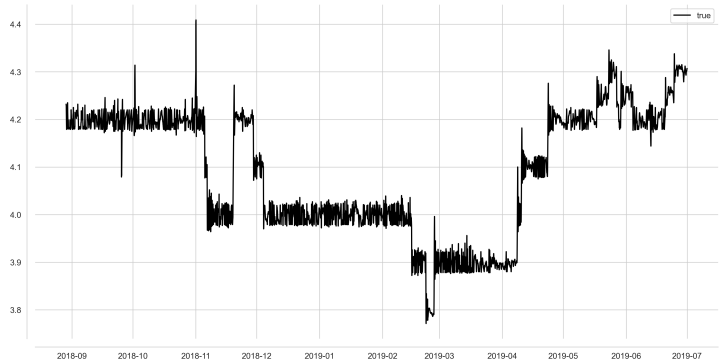
      if type is jump **then**
        $curr\_y\_test[begin\_times[i] : end\_times[i]] += value$

      else ▷ Apply a linear drift to the time series.
        $shift\_per\_timestep \leftarrow \frac{value}{duration}$
        $drift \leftarrow [0, \ldots, duration] \times shift\_per\_timestep$
        $curr\_y\_test[begin\_times[i] : end\_times[i]] += drift$

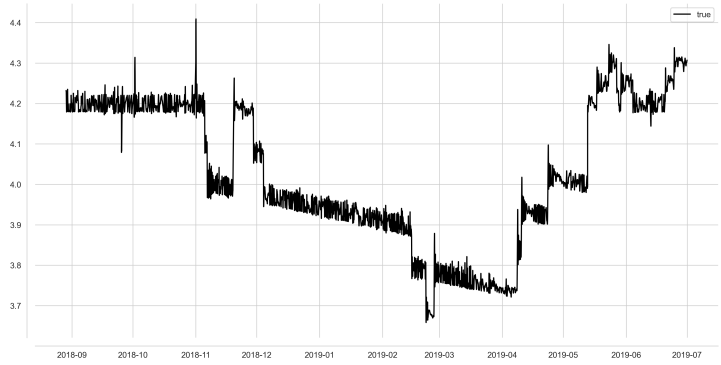
      APPEND($list\_of\_series, curr\_y\_test$) ▷ Add current series to collection.

    return $list\_of\_series$ ▷ List of all adjusted series are returned.

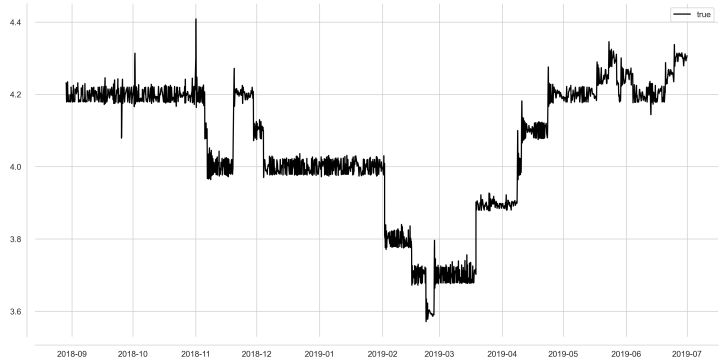
Results of all the experiments are shown in the next sections. First, we display some visual results of a specific series. Afterwards, we will summarise these findings in Section 7.3. Moreover, results are shown in tabular form in Appendix B.1.



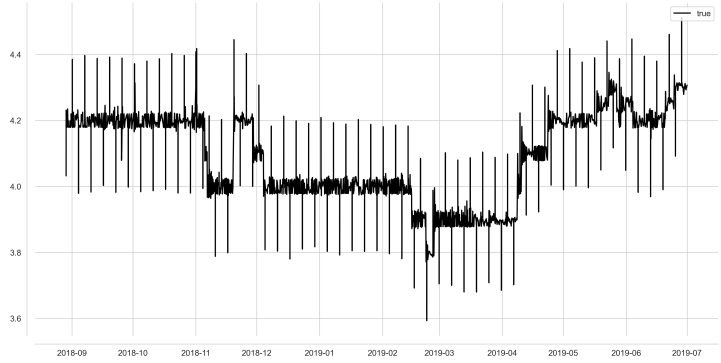
(a) Original time series without added outliers.



(b) Added drift.



(c) Added jump.



(d) Added extremes.

Figure 7.1: Outlier examples in sensor 104OYE for outlier value 0.2. For jump and drift, this is 1 of the 100 created series.

7.2.2 Results multi layer perceptron

For each of the different modelling techniques, we will show the visual results of one specific sensor, one specific outlier category, one specific outlier generation setting and one of the simulated 100 series. Showing all these series is clearly not feasible. Nevertheless, one visual result can already show how the model can function and how realistic quantile boundaries can look.

In Figure 7.2, we display the quantiles and detected outliers of one time series of sensor 104OYE. We added a jump from February 2019 to the middle of March 2019. As we can see, this jump can be easily detected, but some false positives are also present. It is interesting to note that there could actually be some unannotated outliers in this data. This could be the case for the short-lived jump around November 2018, for example. In all successive figures of these plots in the future sections, the same outlier is displayed.

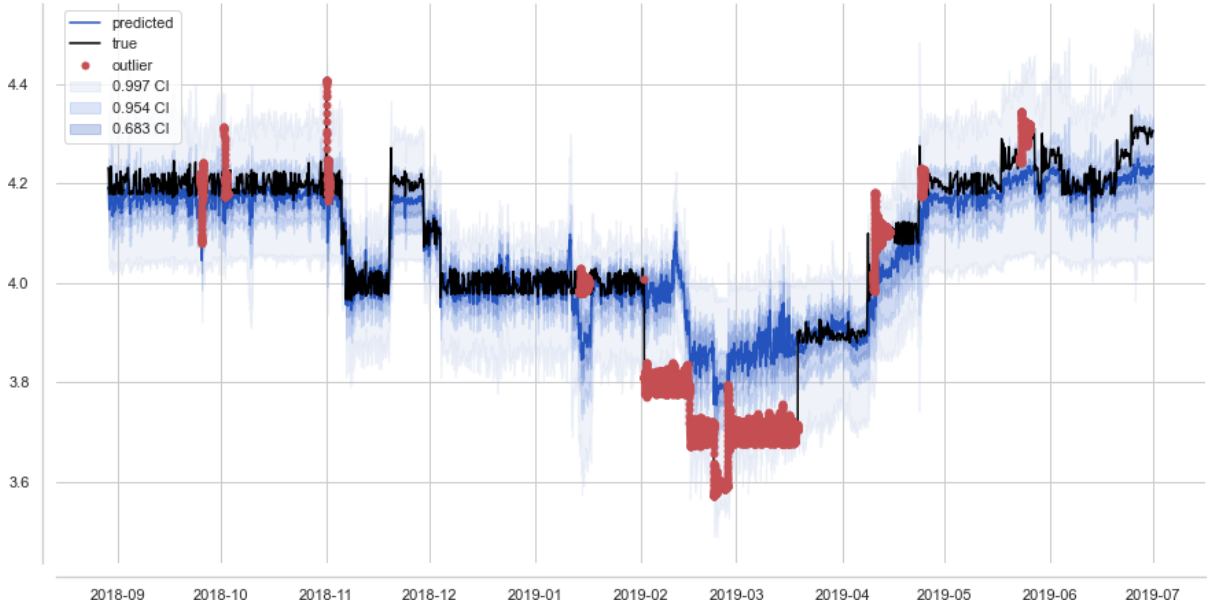


Figure 7.2: Quantile plot of sensor 104OYE, with added jump using the MLP model.

7.2.3 Results perceptron model

The perceptron model shows some decent results in Figure 7.3. It looks similar to the plot of the multi layer perceptron model of Figure 7.2. A difference is that predictions and quantile boundaries look a bit more jagged here. There also seem to be more false alarms.

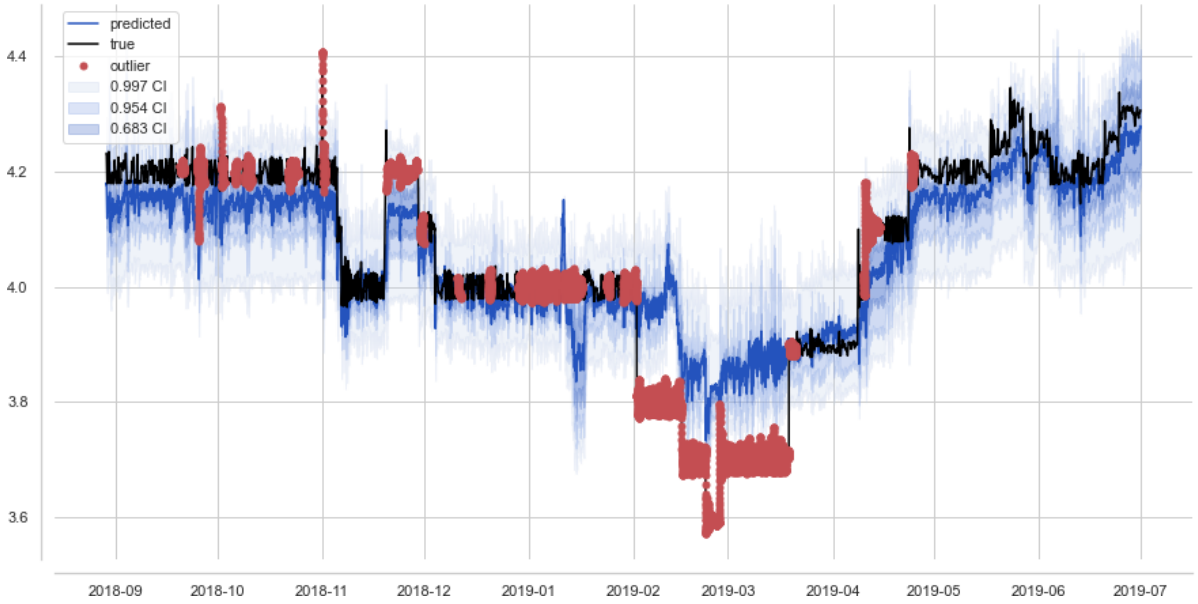


Figure 7.3: Quantile plot of sensor 104OYE, with added jump and perceptron model.

7.2.4 Results RNNs

In Figure 7.4, we see a major issue that apparently can occur when using RNNs. As is clearly visible, the quantiles are very wide. This specific jump could be detected, but it is evident that more subtle outliers (or outliers oriented otherwise) might be harder, or even impossible, to detect.

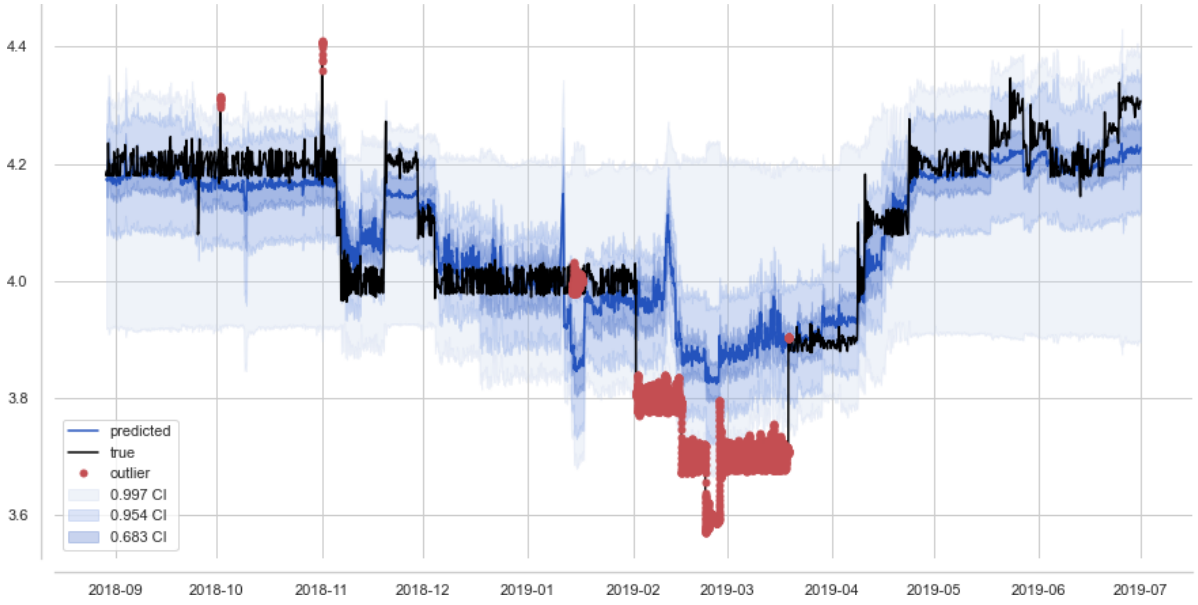


Figure 7.4: Quantile plot of sensor 104OYE, with added jump using the RNN model.

7.2.5 Results quantile regression forests

The quantile regression forest model has major issues, as can be seen in Figure 7.5. The quantile boundaries are very jagged. As a result, many data points are misclassified, even though the added outlier is detected as a whole.

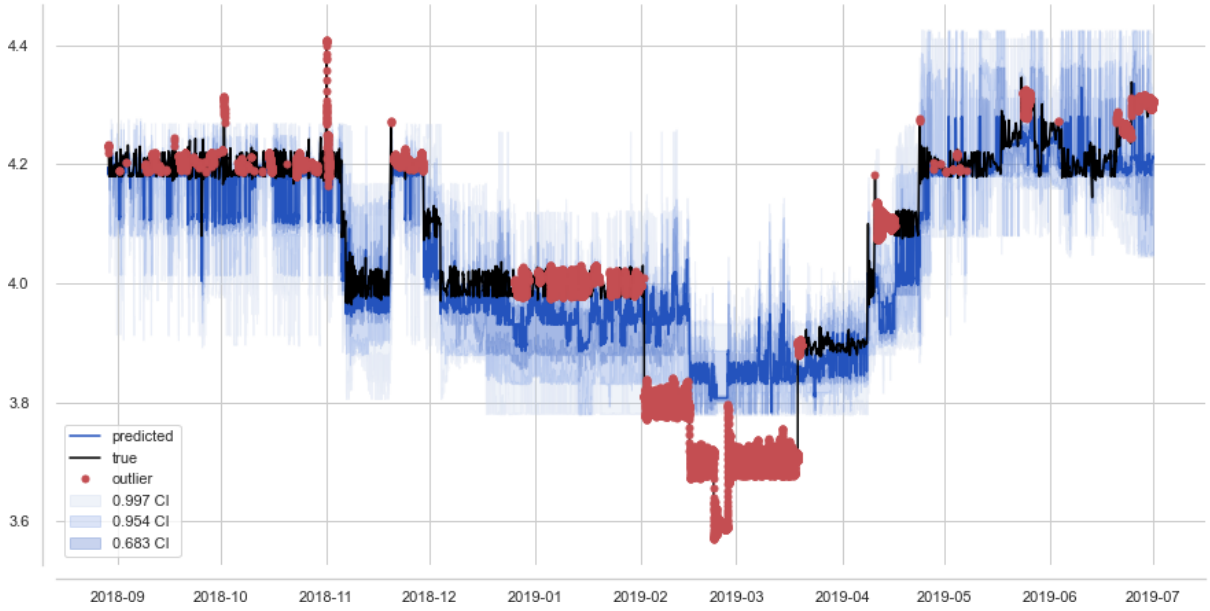


Figure 7.5: Quantile plot of sensor 104OYE, with added jump using the quantile regression forests model.

7.2.6 Results linear regression

The linear regression results depicted in Figure 7.6 look decent. When compared to the other models, though, we clearly see the disadvantage of only having one fixed width for the quantiles throughout the whole data set. Nevertheless, we see that the predicted value for the time series is closely following the actual value most of the time.

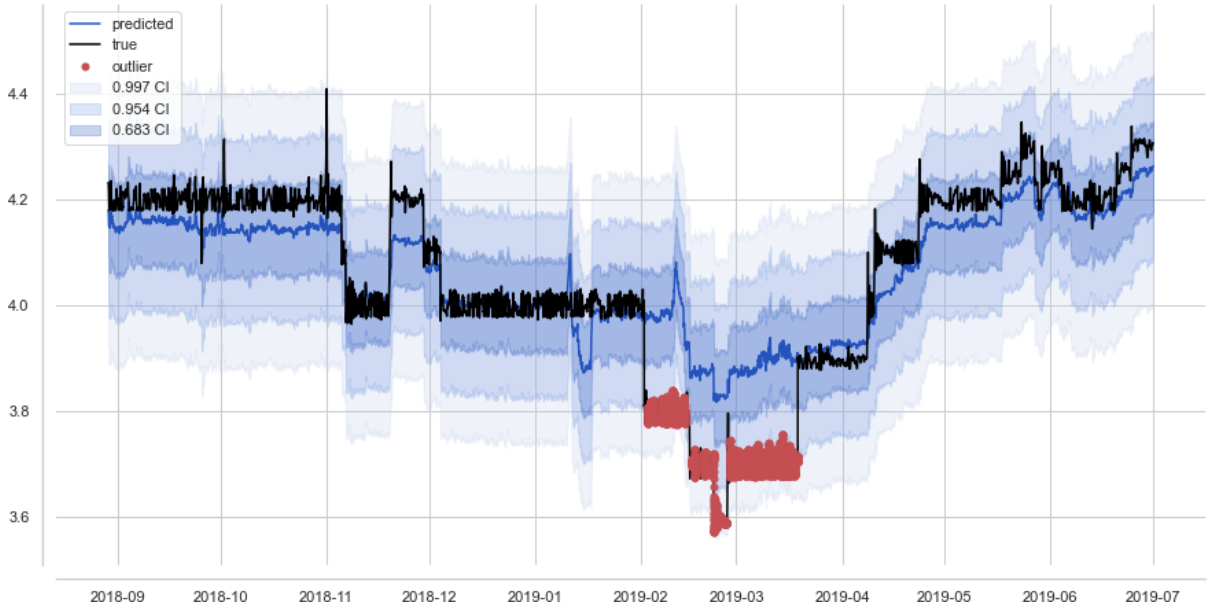
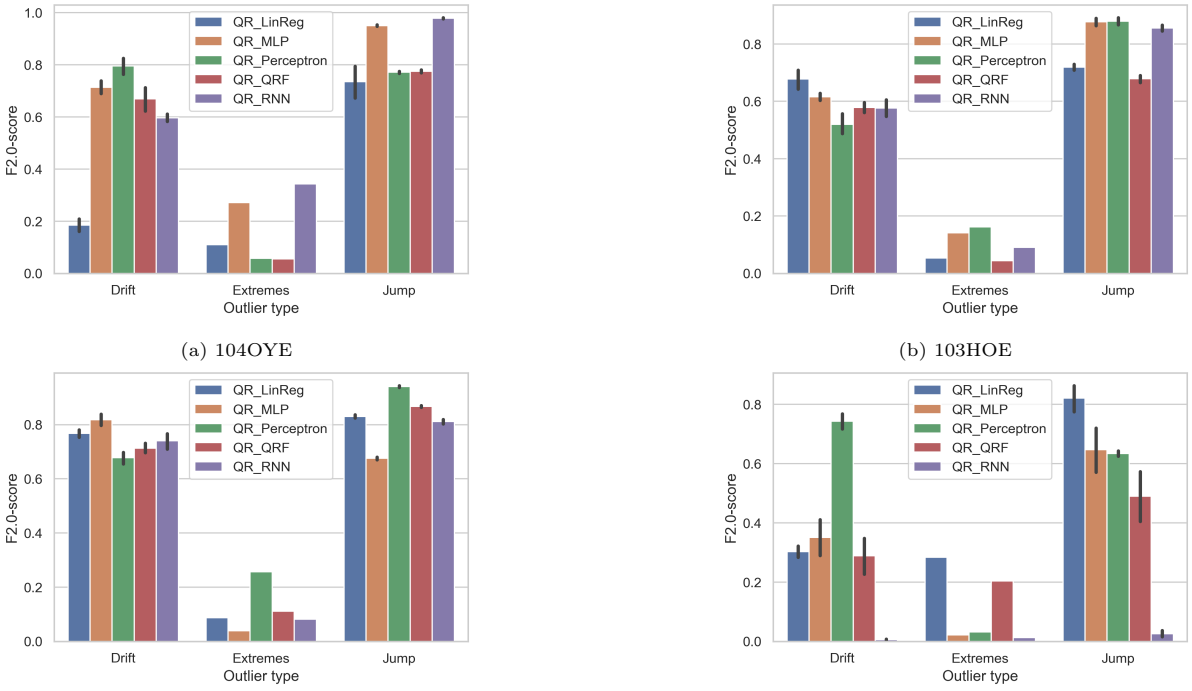


Figure 7.6: Quantile plot of sensor 104OYE, with added jump using the linear regression model.

7.3 Comparison of multivariate modelling techniques

Having gathered and shown all the results of the previous sections, it is yet unclear what conclusions we can draw. To make any claims, we first need to decide which metric we will base the comparisons on. This will be the F2.00-score, as it strikes the right balance between precision and recall for our goals. Recall is more important than precision in our scenario, since the potential impact of a missed outlier will probably be higher than the impact of a false alarm which is sent to a domain expert.

In Figure 7.7, an overview of the F2.00-score results of all models, evaluated on all sensors with all outlier types is presented. Only values for one outlier generation setting are described. This value of 0.2m is realistic for jumps and extremes, according to domain experts. For drift, 0.05m is a more realistic value. Figures 7.8 and 7.9 show the results for values 0.1m and 0.05m. It looks a bit strange that the F2.00-score for extremes is low in all cases. What is key here, is that fewer outliers are added here than in the other categories. In the drift category, 17520 outlier points are added. In the jump category this number is 4320 and when using extremes, 100 different ones are superimposed on the data. The number of total true positives differs greatly per method, which can severely impact precision and thus F2.00-score eventually, as false positives can have a different impact.



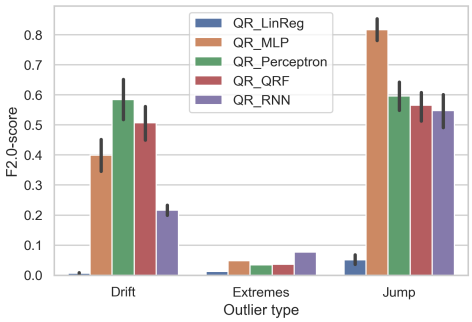
(a) 104OYE

(b) 103HOE

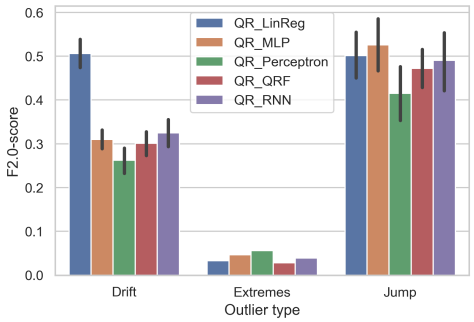
(c) 102BFS

(d) 201D

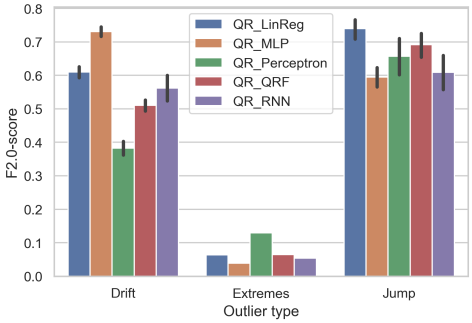
Figure 7.7: Bar plots of model performance on all data sets for outlier value 0.2.



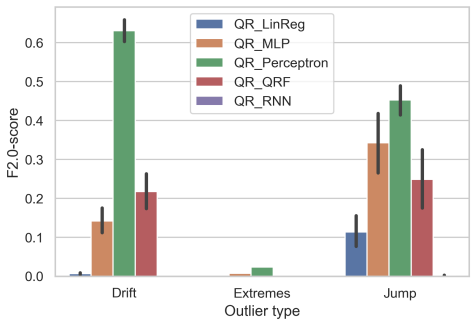
(a) 104OYE



(b) 103HOE

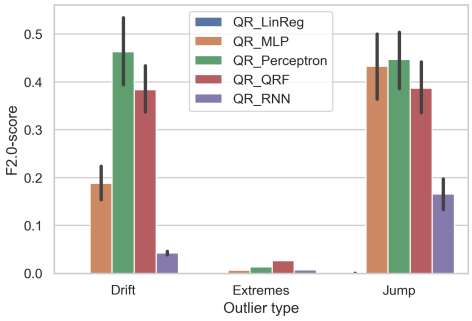


(c) 102BFS

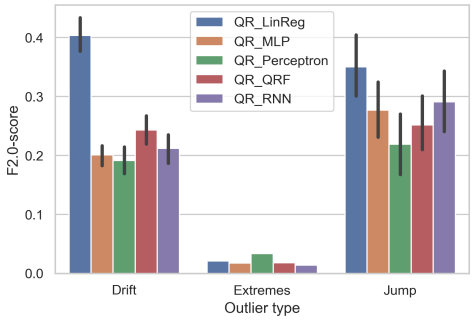


(d) 201D

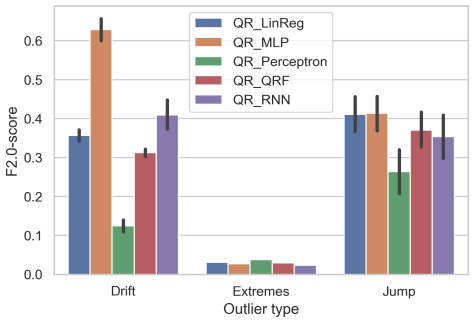
Figure 7.8: Bar plots of model performance on all data sets for outlier value 0.1.



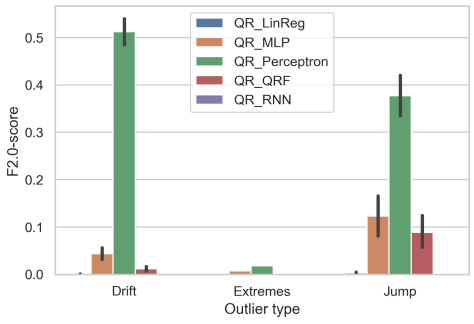
(a) 104OYE



(b) 103HOE



(c) 102BFS



(d) 201D

Figure 7.9: Bar plots of model performance on all data sets for outlier value 0.05.

We see some big differences between the models, but we also note that model performance differs greatly per sensor. To be able to compare these different models statistically, we followed the two-step approach as described by Demšar [9]. First, a Friedman test needs to be performed to check whether there are statistically significant differences between the distributions of the results of the different models. The Friedman test is based on the relative ranking of models on different data sets. The original Friedman test looks at the average ranking when multiple data sets are used. Instead of applying this directly, we used the Friedman Aligned Ranks test, because this is deemed better when few models are being compared, as stated by García et al. [12]. The major difference between these tests is that the Friedman Aligned Ranks tests does not directly combine and average the rankings on multiple data sets, but looks at the ranks of all observations. For each data set, the mean or median classification value (of the different models) is first subtracted from the individual values of the different models. This leads us to the aligned observations. All these observations are ranked, and these ranks are used to determine the outcomes for the different models.

If the Friedman Aligned Ranks test yields a statistically significant difference between the models, we use the Nemenyi test to compare all models pairwise. Demšar [9] recommend this test for the pairwise comparison between models. This enables us to conclude which models perform significantly better than others.

For the multivariate models, we used the Friedman Aligned Ranks test in 8 scenarios. One of them consisted of all data sets, with all added outlier categories. Then, we also divided the data on a sensor basis, which resulted in 4 data sets. The same approach was done to the outlier category, which led us to 3 more data sets. For every outlier, we looked at the specific outlier values to compare: (0.02, 0.05, 0.1, 0.2, 0.3). We did not take the two highest outlier values into account, as they are not realistic.

We define our hypotheses for the Friedman Aligned Ranks test as:

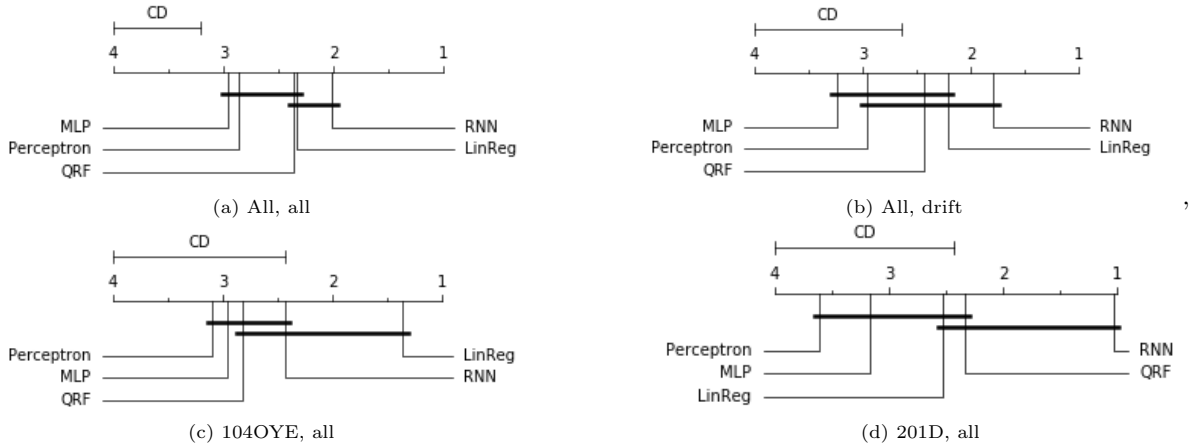
H_0: There is no difference in the distributions of the outcomes of the different models.
H_1: There is a difference in the distributions of the outcomes of the different models.

Sensors	All	All	All	All	102BFS	103HOE	104OYE	201D
Outliers	All	Drift	Jump	Extremes	All	All	All	All
χ^2	14.947	10.556	8.399	4.384	3.287	4.273	11.549	22.646
p-value	0.005	0.032	0.078	0.357	0.511	0.370	0.021	0.000
Conclusion (α=0.05) Reject H_0?	Yes	Yes	No	No	No	No	Yes	Yes

Table 7.2: Results of the Friedman Aligned Ranks test in multivariate modelling.

As we can see, there were four cases where we could reject H_0, which means that there was a statistically significant difference between the model results. We further analysed these cases with the Nemenyi test, to see what models are statistically significant different from one another. Similar to the visualisations created by Demšar [9], we plot the result of this test to get a quick overview of the differences.

As we can see in Figure 7.10, some algorithm types have a significantly different performance. On the horizontal axis, the average ranking of the different algorithms is shown. Algorithms which are connected by a bold line are not differing statistically significantly from each other. The further a model is to the left on the x-axis, the better it is scoring on average. One might expect that a score of 1 is the best, but the rankings are reversed in these results, so that a higher score is better. In all these plots, MLP is significantly different from RNN. In three of them, the perceptron model is also significantly different from RNN.



(a) All, all

(b) All, drift

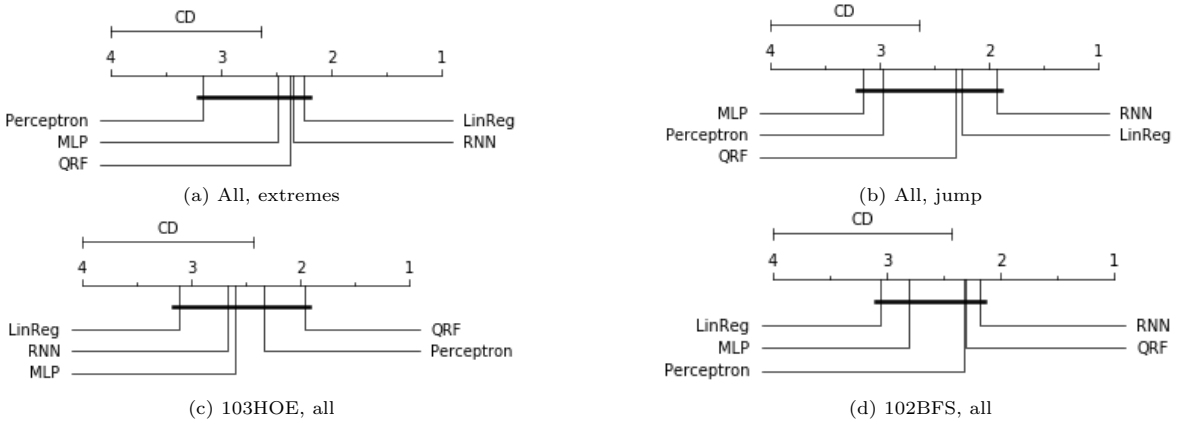
(c) 104OYE, all

(d) 201D, all

Figure 7.10: Nemenyi test results for all cases with a significant difference found by the Friedman Aligned Ranks test.

One note has to be made regarding these graphs. The width of the critical difference (CD) is mainly impacted by the number of data sets n. If the distance on the x-axis between two models $\geq$ CD, there is a statistically significant difference. The higher n, the lower CD will be. All of these tests are with relatively few data sets (where $n = \#sensors \times \#outliertypes \times \#outlier\_values$, which is at highest 60, in the case where all sensors and all outlier types are compared.) As a result, CD will be relatively wide in general.

Although there are not many statistically significant differences between most models, there seems to be a pattern in their ordering. In all 4 cases, MLP or the perceptron model is scoring the best, with the other algorithm as second best. Also, in all cases, RNN scores the worst or the second worst. It is interesting to see if this ordering is also present in the cases where the differences were not statistically significant. These orderings are shown in Figure 7.11.



(a) All, extremes

(b) All, jump

(c) 103HOE, all

(d) 102BFS, all

Figure 7.11: Nemenyi test results for all cases with no significance difference found by the Friedman Aligned Ranks test.

In these plots, it is logical that there is one bold line connecting all models, as there are no statistically significant differences between them. We do not always see the same pattern as observed before. Although MLP and the perceptron model are most of the times among the best performing, this is not always the case. Also, there are some cases where neither of these two models was the best scoring model. We still think, however, that since MLP and the perceptron model are performing decently most of the time, these could be go-to algorithms.

7.4 Synthetic simulation: Univariate results

We now perform the previously described analysis for the univariate models. First, we show the detection performance of the same specific outlier as in 7.2. This was an added jump between February 2019 and the middle of March 2019. In most images, we will now see that this jump does not stand out as much as before, since we are only using the sensor history in these univariate models.

The RNN and quantile regression forests have not been applied to the univariate data, since this became really slow. These models already were relatively slow, but this is especially problematic now, as many more predictions need to be done. These extra predictions are needed because the input of the testing data changes per outlier time series, which was not the case in the multivariate modelling.

The features used for these models were described in Section 5.4.2.

7.4.1 Results multi layer perceptron

For the MLP models, we no longer used a different architecture per sensor. As we do not have to factor in higher or lower correlated time series in anymore and only use the sensor time series itself, it seems like a decent approach to use one basis architecture. This was the same architecture that was used for 104OYE in multivariate MLP modelling, which is a relatively simple architecture, as described in Table 6.2. Figure 7.12 shows that some parts of the added jump can be detected, but this is certainly not the case for the oultier as a whole.

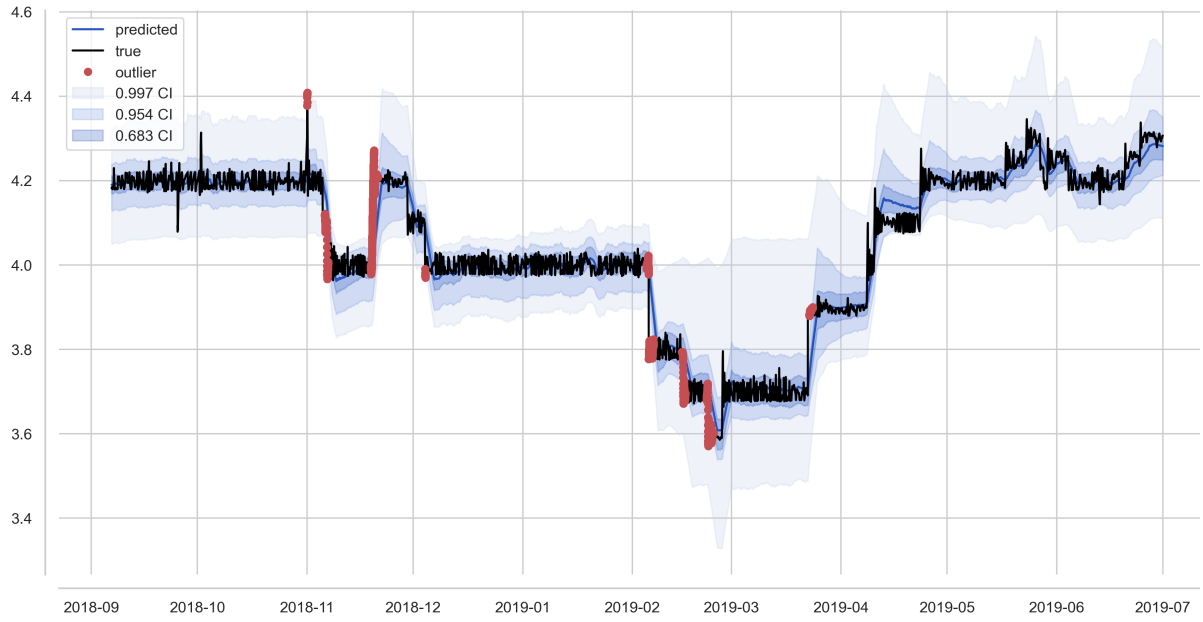


Figure 7.12: Quantile plot of sensor 104OYE, with added jump using the univariate MLP model.

7.4.2 Results perceptron model

The results of this model, displayed in Figure 7.13 look very similar to those of Figure 7.12.

Figure 7.13: Quantile plot of sensor 104OYE, with added jump using the univariate perceptron model.

7.4.3 Results linear regression

The univariate linear regression model shows results which are different from the previous univariate models. This is depicted in Figure 7.14. Just as in the multivariate case, we notice the fixed quantile width. Moreover, we see that the models fails to classify the outlier.

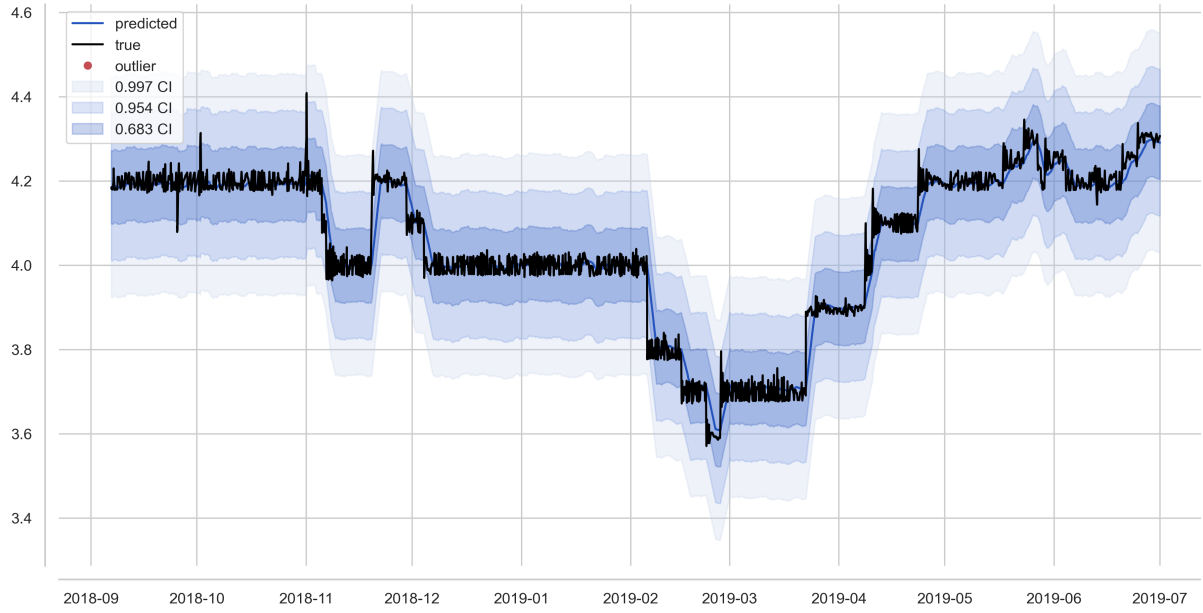


Figure 7.14: Quantile plot of sensor 104OYE, with added jump using the univariate LinReg model.

7.4.4 Results AR

Contrary to all previous models, the AR model does not model quantiles. It only compares the residuals against a threshold to detect outliers. This is seen in Figure 7.15. We see here that the begin and end points of the added outlier can be detected. The period in between can not, though. Also, some other sudden changes in the time series have been classified as outliers.

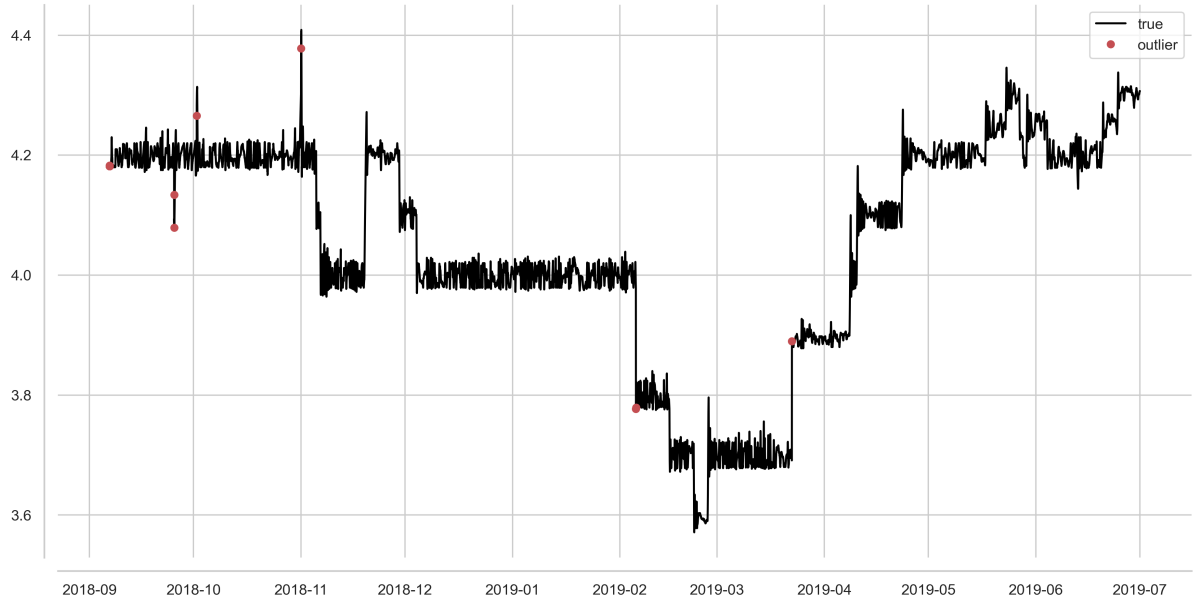


Figure 7.15: Plot of sensor 104OYE, with added jump using the AR model.

7.4.5 Results isolation forest

Similar to the AR model of Section 7.4.4, the isolation forest model does not model quantiles. In Figure 7.16 we see that almost the whole outlier is detected. However, we see that many parts of the time series have been classified as an outlier, which has a huge negative impact on the performance. The detection threshold could be altered, but then we noticed many problems when trying to detect outliers in the extreme values category. It seems that it is not possible to deploy a well-functioning universal threshold for this model.

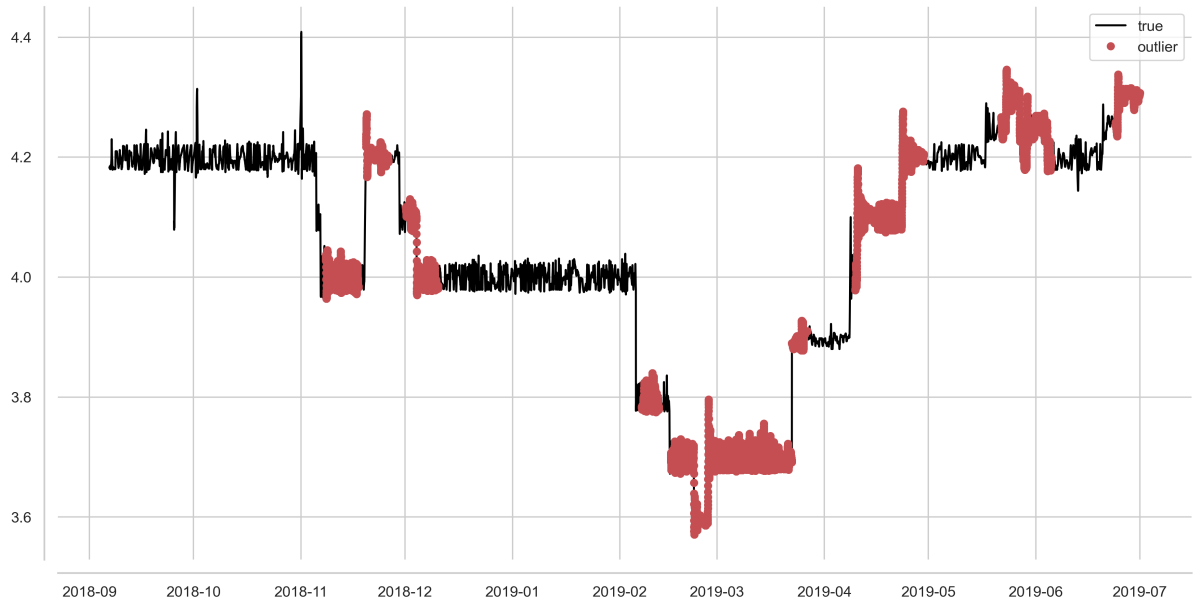
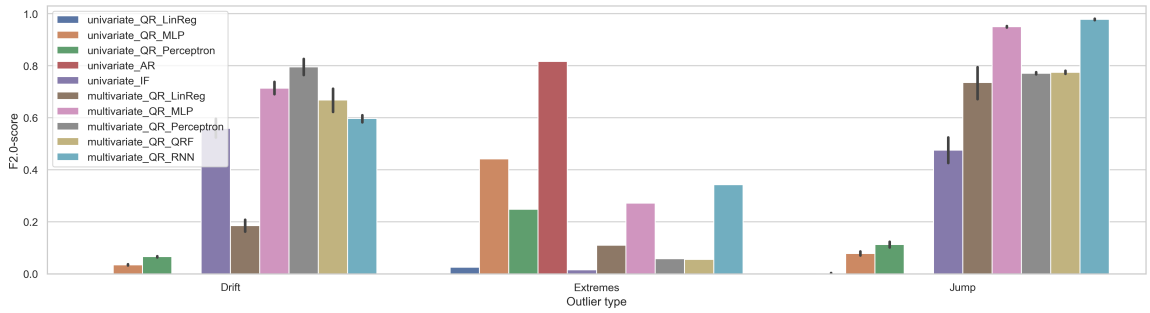


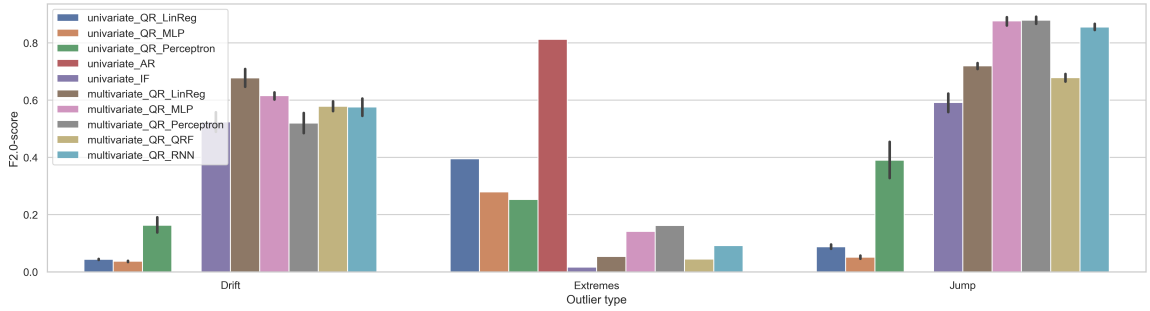
Figure 7.16: Plot of sensor 104OYE, with added jump using the IF model.

7.5 Comparison of univariate and multivariate modelling techniques

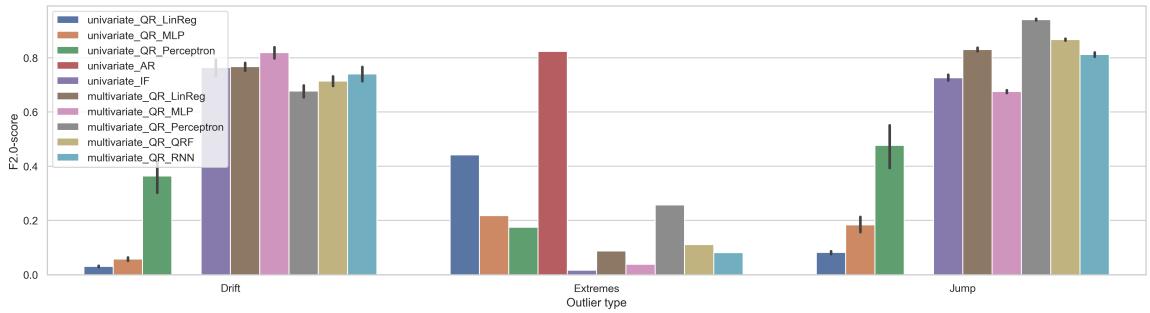
To compare the univariate and multivariate modelling techniques, we first show the F2.00-score results of all models of outlier value 0.2. These results give us much insight in how the different models perform. A visualisation of these results is shown in Figure 7.7.



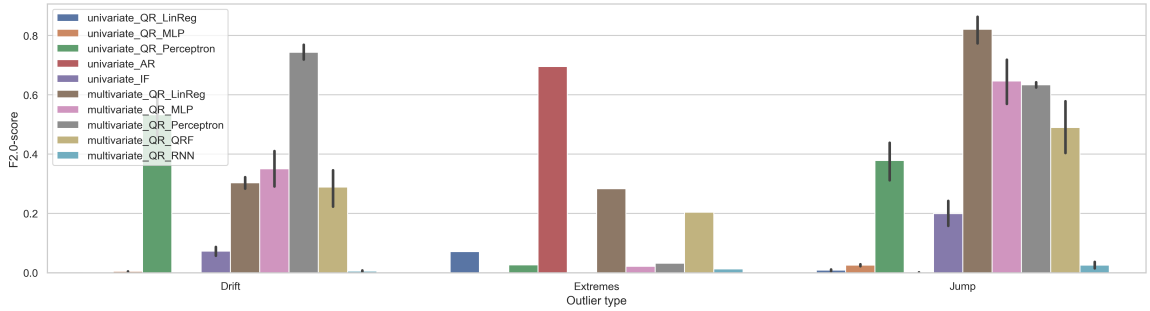
(a) 104OYE



(b) 103HOE



(c) 102BFS



(d) 201D

Figure 7.17: Bar plots of model performance on all data sets, univariate and multivariate combined for outlier value 0.2.

In these bar plots, we see some clear differences. To make solid claims about the observed differences, we use the Friedman Aligned Ranks test again. We now include both the univariate and multivariate results to test if there are statistically significant differences in the distributions.

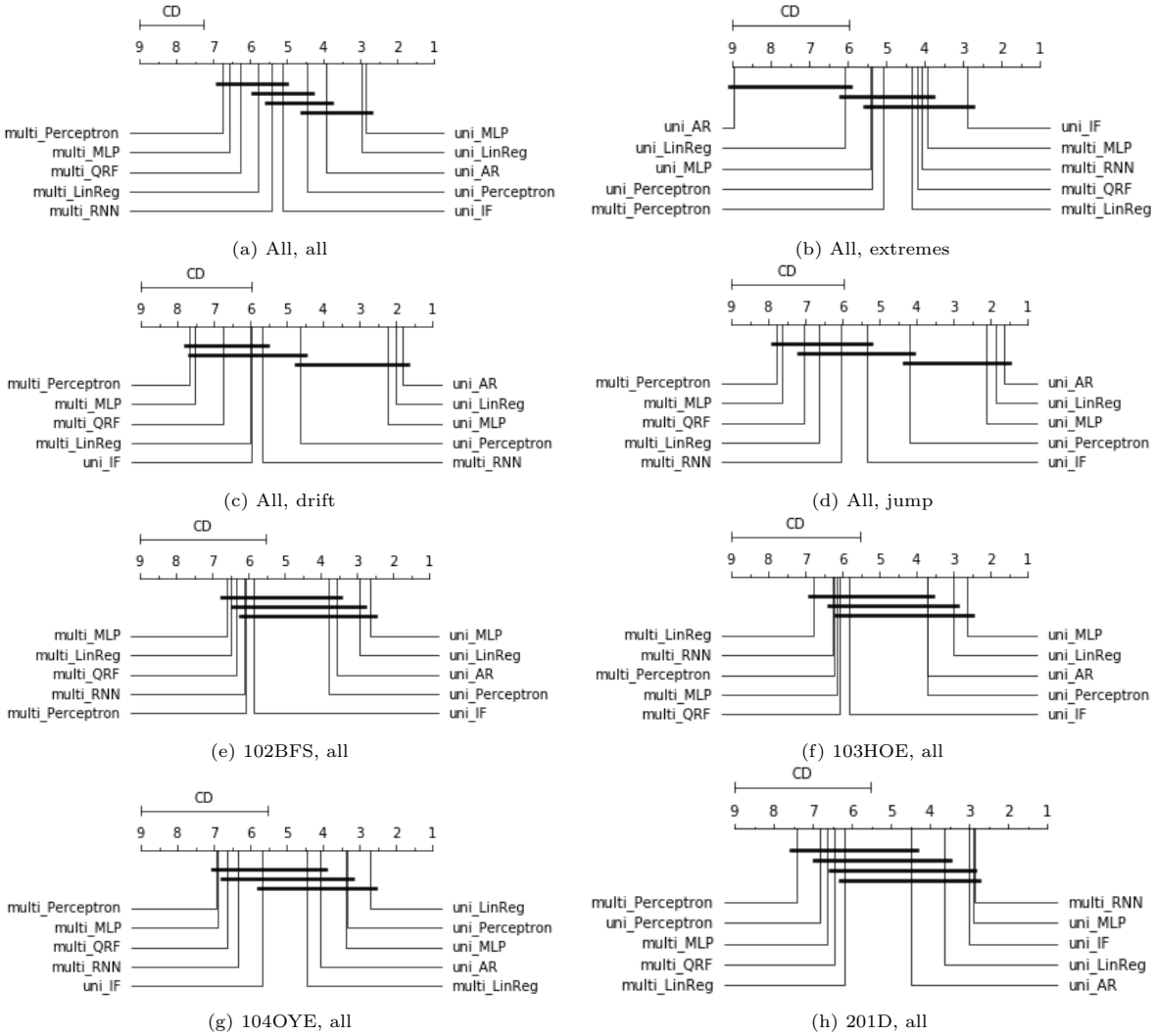
Sensors	All	All	All	All	102BFS	103HOE	104OYE	201D
Outliers	All	Drift	Jump	Extremes	All	All	All	All
χ^2	118.936	100.000	115.084	60.752	38.274	36.949	39.365	51.748
p-value	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Conclusion (α=0.05) Reject H_0?	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 7.3: Results of the Friedman Aligned Ranks test of multivariate and univariate modelling experiments.

We notice that all these distributions are statistically significantly different from each other. This is expected, as we compare many models which have very different performances as is clearly visible in Figure 7.17.

We now perform the Nemenyi test to see which models are differing statistically significantly from one another. The results are shown in Figure 7.18. In almost all cases, a multivariate model is outperforming univariate ones. A notable exception is the extreme outlier category, where AR seems to perform exceptionally well. This is due to the fact that AR almost works like a persistence model, as explained in 6.2.1. Since this is the case, a large sudden change (like the way in which we implemented extremes) is easily detected.

It is notable that in Figure 7.18a we see that the 5 best scoring models are all the multivariate models. This is because they score better in drift and jump detection scenarios. If we want one model to detect



(a) All, all

(b) All, extremes

(c) All, drift

(d) All, jump

(e) 102BFS, all

(f) 103HOE, all

(g) 104OYE, all

(h) 201D, all

Figure 7.18: Nemenyi test results for all cases: univariate and multivariate combined. In all results, significance difference was found by the Friedman Aligned Ranks test.

all these outlier types, then a multivariate model like one of the MLP models is probably the best choice. Depending on the kind of extremes that are possible in practice, however, univariate approaches might also be very interesting. The AR approach might be too simple in most cases, but in practice it can turn out to be beneficial as shown here. It might also be the case that our added extreme values are too simplistic, but that depends on the specific kind of outlier that is to be detected.

7.6 Practical impact

In this section, we will assess the practical impact of our models and make some practical suggestions. In previous sections we have shown the multivariate MLP algorithm to be very interesting for our aims. An advantage of this model is that it should generalise to many different kinds of outliers, not only to the synthetic outlier categories we created. We will show this in a time series (sensor 108HOL) which we know to have outliers in it. These outliers were annotated by domain experts. In Figure 7.19, we can see an outlier is present around October 2018 and is detected nicely. This is a different outlier than our synthetic ones but can still be detected.

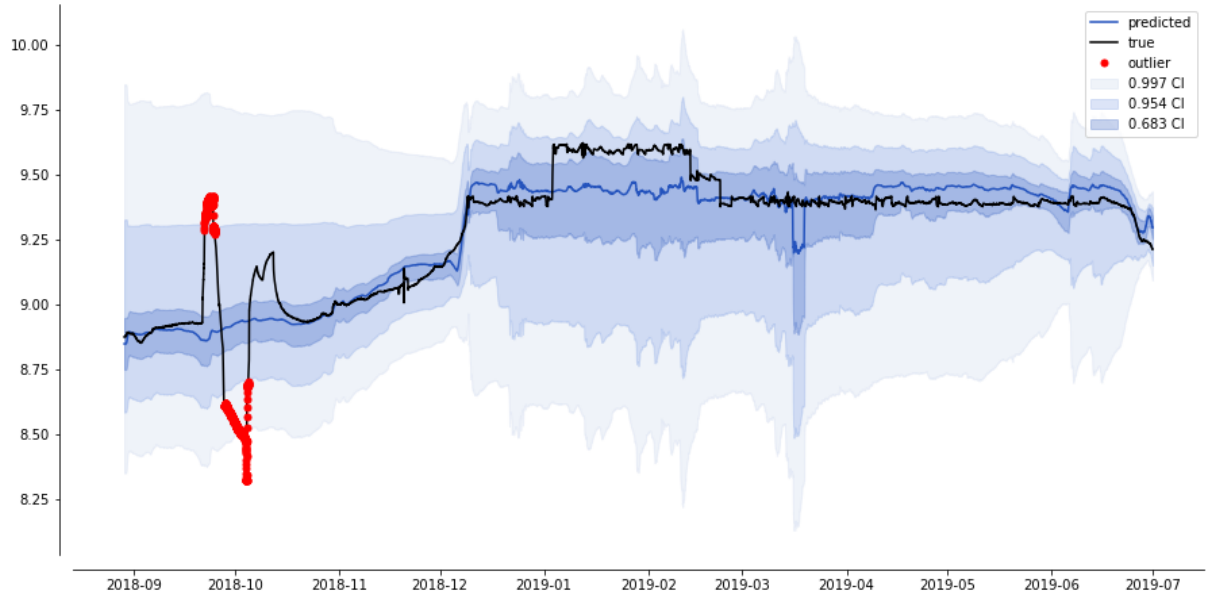


Figure 7.19: Outlier detection plot of 108HOL modelled by the multivariate MLP model. No outliers were added. Although the already present outlier around October 2018 can not be detected fully, it still can be detected well.

It is also interesting to look at the validation loss of the trained model and the correlation between the different underlying time series. Validation loss is a measure of model performance and takes the quantiles into account. This makes it more useful here than an R^2-score, for example, as this only takes the mean prediction into account. We examine results from only the MLP model to keep this section more succinct. We also examine the correlation of the underlying time series, to see if we can find a relation between correlation and validation loss. These results are shown in Table 7.4. There seems to be a trend here. The lower the least correlated time series value, the higher the validation loss. This makes sense intuitively, as correlated time series are likely to improve model performance. If we look at the plots from Appendix A, we see a similar result. 104OYE seems to be modelled very well whereas 102BFS has some obvious problems, resulting in (almost) straight lines. It thus seems to be the case that correlated time series are a requirement for performant outlier detection.

Sensor	Validation loss	Least correlation
102BFS	1.5267	0.357
103HOE	0.8703	0.691
104OYE	0.1796	0.816
201D	0.9459	0.682

Table 7.4: Validation loss in multivariate MLP experiments and correlation for sensors. Least correlation indicates the correlation of lowest correlated sensor of the 4 selected (highest) ones, as described in Section 5.2.

Another interesting practical result is the number of time steps needed to find drifts. The data used for these findings is described in Section B.1.1. We can see that almost no drifts were missed at all, even in the 0.05m outlier generation setting. We do see, however, that the number of false positives is lower there than in the easier generation settings. False positives could also contribute to this. Furthermore, we see that we generally need about 3000 time steps, which corresponds to approximately one month, before a drift is detected. Domain experts stated that periodical checks for the occurrence of drift are normally performed yearly, so improvements seem possible by using our models. Moreover, jump and extremes values of 0.2m are detected nicely, as we see in Section B.1.1.

Chapter 8

Discussion

Chapter 7 has shown that some outlier types can be detected better than others. Domain experts stated that jump values and extreme values of 0.20m are reasonable in real life. We can derive from Figure 7.17 that the performance of the best performing models seems satisfactory in many cases. The domain experts also stated that a drift is generally in between 0.05m-0.10m throughout a year. This roughly corresponds to our two categories of outlier values 0.02 and 0.05, since we apply drift with a duration of half a year. When examining the results of Figure 7.9, we see that outliers for these values can not really be found well. Based on the F2.00 values, it seems the case that our models are not performing that well. However, this judgement is (too) harsh, because every value in the range of an added outlier is labelled as such. This even counts for the beginning of drifts. Thus, the F2.00 values underestimate our results and it was not surprising that Section 7.6 could show practical impact and possibilities for drift detection by using our models. Drift can be detected in approximately one month, which leads to improvements over a manual periodical check.

Examining the results of the extreme value scenarios a bit closer, it seems that the F2.00-scores for the multivariate models are relatively low. If we inspect the recall measures more closely, however, we see that the multivariate MLP, perceptron and quantile linear regression model have high recall in most of these cases. The precision is low on the contrary. This is due to the occurrence of a combination of some false positives and a relatively low number of true positives. In practice, it might not be problematic if some false alarms are sent to domain experts. So, our performance is still satisfactory.

Another remark is that visual results and F2.00 scores do not always result in similar conclusions. It can be the case that a F2.00 score may not indicate many problems, but a visual look at the quantile plot raises some concerns. An example is the quantile regression forest plot of Figure 7.5. This result does not look convincing, but the F2.00 scores of QRF in Figures 7.7a, 7.8a and 7.9a are decent. This is also visible in the collection of plots of Appendix A, which can be compared with the F2.00 scores of Appendix B. This leads us to believe that it is paramount that the F2.00 scores should always be viewed alongside with quantile plots to ensure decent performance.

We discovered that the tuning of models is not always straightforward. It was assumed that for a certain model type, like MLP, one general model specification should be enough. It turned out that this was not the case. This is problematic for applying these approaches in practice, as a model needs to be tuned for every sensor. On the other hand, when we take the perceptron method which has a constant model architecture, we still see decent performance. So, this might not be that problematic. The size of the issue depends on the specific application and on how many different time series are present.

Our unsupervised models have a key advantage that they can detect multiple kinds of outliers. Everything that is different from what is expected, is classified as an outlier. This did not only work for our synthetic outlier categories, but also for the already present outlier in Section 7.6.

Chapter 9

Conclusion and future work

9.1 Conclusion

In this work, we applied multivariate and univariate real-time outlier detection models in unlabelled water height time series.

This work has a number of contributions. The largest one is the systematic comparison of algorithms and the easily parametrisable synthetic validation scenarios which were constructed in cooperation with domain experts. This results in the comparison of multivariate models between themselves, and also in the comparison of multivariate and univariate models. Another contribution is the imputation benchmark for systematically comparing imputation methods. Also, we established a thoroughly researched pipeline approach to detect outliers.

The multivariate modelling approach is not applicable for all sensors (like 102BFS). Some sensor predictions clearly work better than others. The correlation between sensors need to be high enough to create an accurate model, as was shown in Section 7.6.

An interesting discovery was that there is a whole category of models which can work for outlier detection in general, but are not helpful in this multivariate case. These were described in Section 6.2.1, 6.3.5 and 6.4. We did not find previous indications that these models would not really work in our multivariate experiments, so we think this insight is fairly interesting. In the univariate case these outlier detection models still work. A notable finding was the result of the AR model of Section 6.2.1, which worked exceptionally well for the extremes outlier category. This category has some issues, however. It is an artificial scenario in which the univariate models like AR score relatively high. Since only sudden changes in values are detected, we can detect extremes here, but can not generalise to realistic behaviour. As shown, it fails to detect jumps and drifts efficiently.

We now return to the research questions of Section 2.1. The first one concerned the comparison of multivariate modelling approaches for real-time outlier detection in unlabelled and unvalidated time series data. Chapter 7 (and more specifically the results of the Friedman Aligned Ranks tests and Nemenyi tests from Sections 7.3 and 7.5) helps us in answering these questions. The quantile regression forest approach is relatively slow, does not perform well and gives us poor visual results. The RNN models performed worse than expected. Linear regression performed relatively decently, although the fixed quantile width remains an issue. In the end, we can state that the MLP and the perceptron models performed the best overall. The results of the latter were alike to the results of the MLP.

Another aim of our research was to determine whether multivariate modelling approaches improve performance compared to univariate approaches. Multivariate models perform better in most of the cases, as was shown in Section 7.5. Multivariate modelling behaviour looks better visually as well.

Extreme values are a notable exception, however. Univariate modelling performed better in that outlier category, although our added extreme values are slightly simplistic. An advantage of the univariate modelling is that it is always applicable and does not depend on the availability of correlated time series, which was one of the reasons why the comparison with multivariate methods was made.

9.2 Future work

Our research was aimed at comparing different models in realistic outlier generation settings. To get a better overview how well these different models work in practice, however, it is recommended that a pilot program is carried out to test the performance in a more practical setting. For example, different models can be assessed in a production environment for the period of one year on a large set of sensors. Interesting insights could be derived in this practical pilot program, such as how quickly different kinds of outliers can be detected in practice.

It is an interesting idea to use different models to approach different outlier categories. For example, combining the results of an AR model and a multivariate MLP model could work to detect extreme values, jumps, and drifts.

Furthermore, there were some candidate research questions which remain open for future research. An interesting research subtopic regards determining outlier causes. This was declared to be out of scope for our work. An outlier can be caused by multiple factors (i.e. an erroneous measurement, drift, noise, technical equipment failure, etc.). Different kinds of outliers might require different means of alleviation. It is of interest to determine these different causes with additional techniques.

Another interesting subtopic concerns propagating sensor errors. If a sensor malfunctions, this will not only affect its own predictions, but will affect all other sensor predictions that make use of the values of this malfunctioning sensor as a predictor variable as well. Since we have many sensors which are relatively close to one another, it makes sense to use values from other sensors as predictor variables. But, when one sensor is used in multiple other predictions, many predicted values can be wrong because just one sensor is not behaving correctly. This might be detected by examining the correlations between the underlying time series in a multivariate case. It could be the case that, in a propagating error scenario, the time series with the actual outlier itself will have the lowest correlation with the others. Further research is needed to make accurate claims about this phenomenon.

Another venue of future work is related to sensor selection. In our work, there were no (hydrological) checks in place to see if it makes sense that multiple sensor time series are correlated. So, it could be the case that some models are based on sensors which would not be described as logical or sensible by domain experts. This can lead to a less convincing and understandable modelling procedure in the eyes of domain experts. It is possible to create extra checks, which can be based on catchment areas, on physical distance between sensors or on the graph structure of the connectivity tree of the weir, for example.

A similar idea is to perform more experiments in feature selection, which was mentioned in Section 5.2. More specifically, it can be very interesting to compare (or even combine) a filter and a wrapper approach for sensor selection.

An insight was derived in discussion with the domain experts. If a multivariate model is performing poorly, it can be the case that the underlying time series that are being used for predictions have issues. In this sense, a poorly performing model can also be used for a manner of detection. More research into this avenue is needed.

Moreover, it is interesting to see how well these models will perform on other data sets. Other data sets in possibly other domains can have other (cor)relations in the data. The models should probably still be applied to a multivariate case with multiple sources of correlated data, just like is the case in our

work. We can see many cases where this could be useful. These models can improve data validation in a large factory or industry terrain, for example.

Currently, many novel algorithms and algorithm variations get introduced in the field of machine learning and data science. It is infeasible to create models with all these virtually unlimited algorithm variations. One interesting recent contribution to the field of neural networks is the Transformer model by Vaswani et al. [47], which is a deep learning model invented for the use in natural language processing. This model makes use of an attention mechanism, which uses weights in such a way that long-term dependencies can be easily modelled. Although this model was not created for time series modelling purposes, the concept has been adapted to time series by Qin et al. [38] and Lai et al. [24] among others. Nevertheless, standardisation of this algorithm approach and standard implementation in neural network libraries has not yet been done. Research into this highly interesting area was considered out of scope of this project and could be carried out in future work.

Bibliography

[1] C. C. Aggarwal. Outlier analysis. In *Data mining*, pages 237–263. Springer, 2015.

[2] M. Amer, M. Goldstein, and S. Abdennadher. Enhancing one-class support vector machines for unsupervised anomaly detection. In *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, pages 8–15. ACM, 2013.

[3] I. Bae and U. Ji. Outlier detection and smoothing process for water level data measured by ultrasonic sensor in stream flows. *Water*, 11(5):951, 2019.

[4] G. E. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2):211–243, 1964.

[5] C. Chatfield. *The analysis of time series: an introduction*. Chapman and Hall/CRC, 2003.

[6] N. Chinchor and B. Sundheim. MUC-5 evaluation metrics. In *Proceedings of the 5th conference on Message understanding*, pages 69–78. Association for Computational Linguistics, 1993.

[7] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[8] S. F. Crone and N. Kourentzes. Feature selection for time series prediction–a combined filter and wrapper approach for neural networks. *Neurocomputing*, 73(10-12):1923–1936, 2010.

[9] J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research*, 7 (Jan):1–30, 2006.

[10] Z. Ding and M. Fei. An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. *IFAC Proceedings Volumes*, 46(20):12–17, 2013.

[11] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):44, 2014.

[12] S. García, A. Fernández, J. Luengo, and F. Herrera. Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044–2064, 2010.

[13] O. Gencoglu, M. van Gils, E. Guldogan, C. Morikawa, M. Süzen, M. Gruber, J. Leinonen, and H. Huttunen. HARK Side of Deep Learning–From Grad Student Descent to Automated Machine Learning. *arXiv preprint arXiv:1904.07633*, 2019.

[14] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han. Outlier detection for temporal data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 26(9):2250–2267, 2013.

[15] M. A. Hall. Correlation-based feature selection for machine learning. *PhD Thesis*, 1999.

[16] D. M. Hawkins. *Identification of outliers*, volume 11. Springer, 1980.

[17] D. J. Hill and B. S. Minsker. Anomaly detection in streaming environmental sensor data: A data-driven modeling approach. *Environmental Modelling & Software*, 25(9):1014–1022, 2010.

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

[19] R. Jozefowicz, W. Zaremba, and I. Sutskever. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pages 2342–2350, 2015.

[20] A. Khosravi, S. Nahavandi, D. Creighton, and A. F. Atiya. Comprehensive review of neural network-based prediction intervals and new advances. *IEEE Transactions on neural networks*, 22(9):1341–1356, 2011.

[21] R. Koenker and K. F. Hallock. Quantile regression. *Journal of economic perspectives*, 15(4):143–156, 2001.

[22] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324, 1997.

[23] M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37 (2):233–243, 1991.

[24] G. Lai, W.-C. Chang, Y. Yang, and H. Liu. Modeling long-and short-term temporal patterns with deep neural networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 95–104, 2018.

[25] B. Lamrini, A. Gjini, S. Daudin, P. Pratmarty, F. Armando, and L. Travé-Massuyès. Anomaly detection using similarity-based one-class svm for network traffic characterization. In *DX@ Safeprocess*, 2018.

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

[27] C. Leigh, O. Alsibai, R. J. Hyndman, S. Kandanaarachchi, O. C. King, J. M. McGree, C. Neelamraju, J. Strauss, P. D. Talagala, R. D. Turner, et al. A framework for automated anomaly detection in high frequency water-quality data from in situ sensors. *Science of The Total Environment*, 664:885–898, 2019.

[28] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

[29] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[30] H. Lütkepohl. *New introduction to multiple time series analysis*. Springer Science & Business Media, 2005.

[31] H. Lütkepohl. Vector autoregressive models. In *Handbook of Research Methods and Applications in Empirical Macroeconomics*. Edward Elgar Publishing, 2013.

[32] J. Ma and S. Perkins. Time-series novelty detection using one-class support vector machines. In *Proceedings of the International Joint Conference on Neural Networks, 2003.*, volume 3, pages 1741–1745. IEEE, 2003.

[33] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

[34] D. Masters and C. Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

[35] N. Meinshausen. Quantile regression forests. *Journal of Machine Learning Research*, 7(Jun):983–999, 2006.

[36] L. Perelman, J. Arad, M. Housh, and A. Ostfeld. Event detection in water distribution systems from multivariate water quality time series. *Environmental science & technology*, 46(15):8212–8219, 2012.

[37] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko. A review of novelty detection. *Signal Processing*, 99: 215–249, 2014.

[38] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*, 2017.

[39] C. Ranjan, M. Reddy, M. Mustonen, K. Paynabar, and K. Pourak. Dataset: rare event classification in multivariate time series. *arXiv preprint arXiv:1809.10717*, 2018.

[40] N. Reunanen, T. Räty, J. J. Jokinen, T. Hoyt, and D. Culler. Unsupervised online detection and prediction of outliers in streams of sensor data. *International Journal of Data Science and Analytics*, pages 1–30, 2019.

[41] F. Rodrigues and F. C. Pereira. Beyond expectation: Deep joint mean and quantile regression for spatio-temporal problems. *arXiv preprint arXiv:1808.08798*, 2018.

[42] S. Sadik and L. Gruenwald. Research issues in outlier detection for data streams. *Acm Sigkdd Explorations Newsletter*, 15(1):33–40, 2014.

[43] P. D. Talagala, R. J. Hyndman, C. Leigh, K. Mengersen, and K. Smith-Miles. A feature-based framework for detecting technical outliers in water-quality data from in situ sensors. *arXiv preprint arXiv:1902.06351*, 2019.

[44] P. D. Talagala, R. J. Hyndman, and K. Smith-Miles. Anomaly detection in high dimensional data. *arXiv preprint arXiv:1908.04000*, 2019.

[45] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.

[46] S. Van Buuren and K. Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations in R. *Journal of statistical software*, pages 1–68, 2010.

[47] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[48] R. Versteeg and B. de Graaff. Valdidatieplan waterkwantiteitsmetingen. STOWA 2009-20, 2009.

[49] Western Electric Company. *Statistical quality control handbook*. Western Electric Company, 1956.

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

[51] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

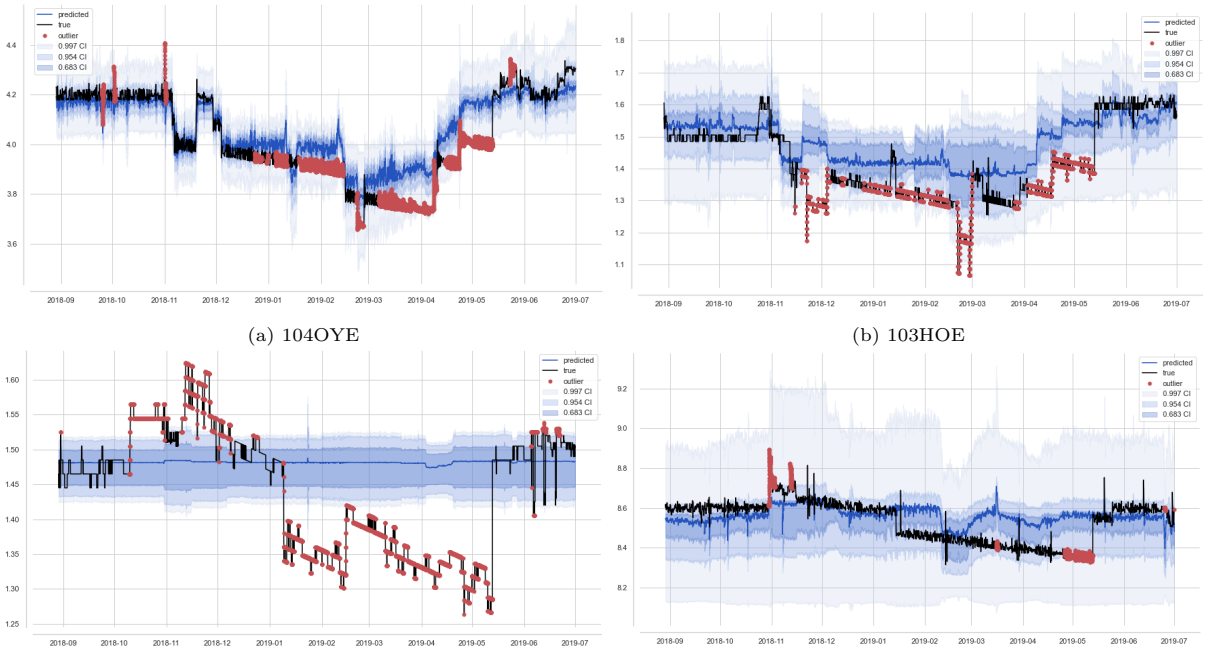
[52] I.-K. Yeo and R. A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000.

[53] L. Zhu and N. Laptev. Deep and confident prediction for time series at uber. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 103–110. IEEE, 2017.

Appendix A

Multivariate MLP quantile plots

To give more insight into the (visual) results of our multivariate MLP algorithm, we show the results of this algorithm on all sensors with outlier generation setting of 0.2 of all different outliers in Figure A.1, A.2 and A.3. It is clearly visible that some sensors (104OYE, 103HOE) are modelled much better than some others (102BFS, 201D).



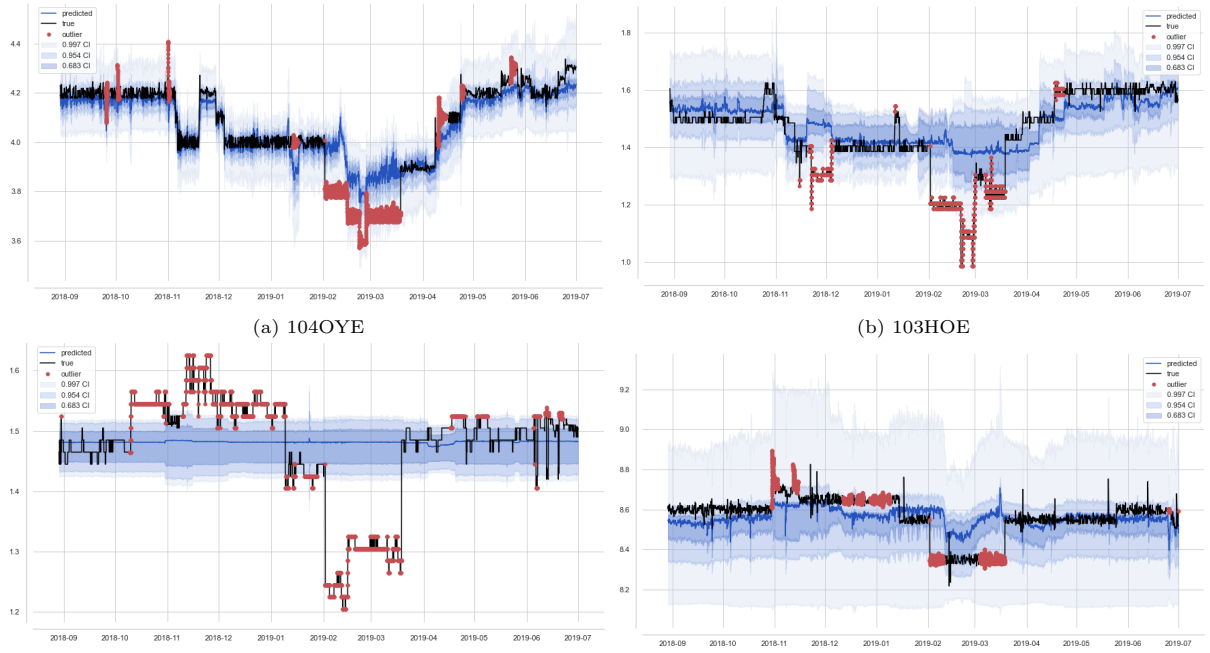
(a) 104OYE

(b) 103HOE

(c) 102BFS

(d) 201D

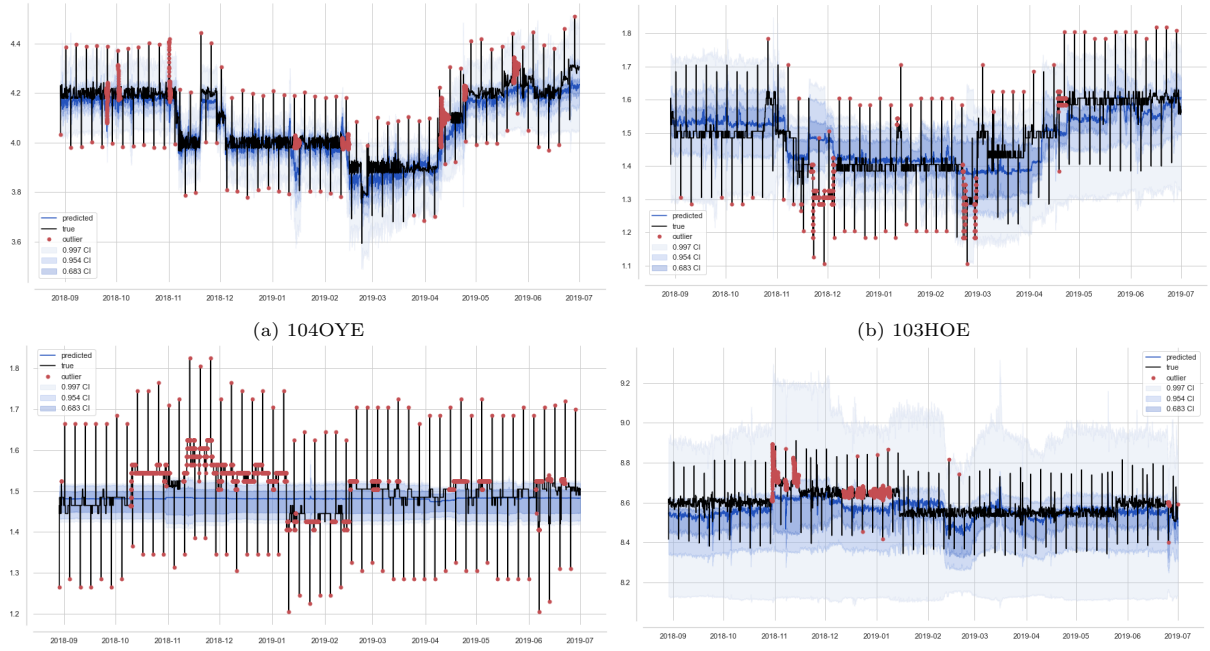
Figure A.1: Quantile detection plots of drift outliers on all data sets.



(a) 104OYE (b) 103HOE

(c) 102BFS (d) 201D

Figure A.2: Quantile detection plots of jump outliers on all data sets.

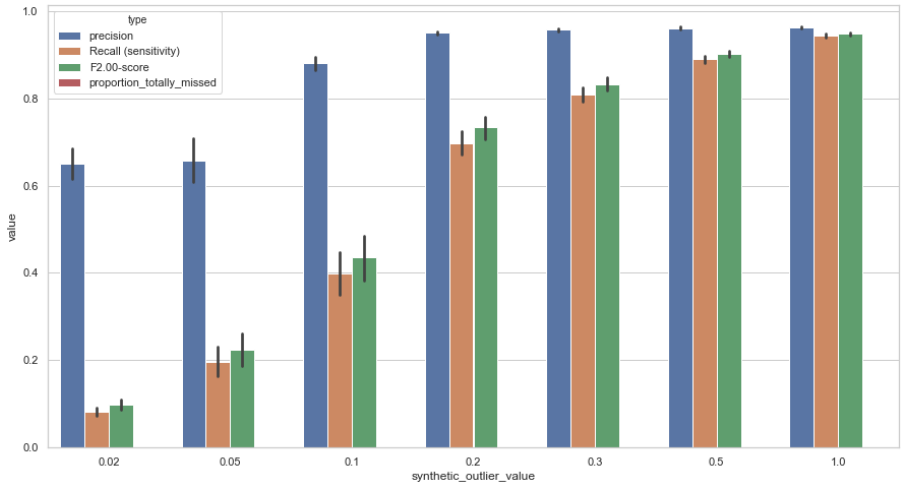


(a) 104OYE (b) 103HOE

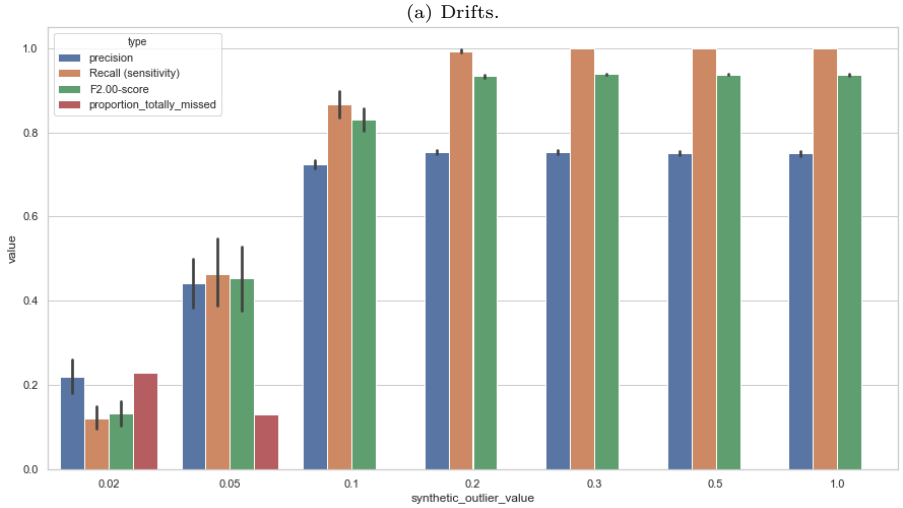
(c) 102BFS (d) 201D

Figure A.3: Quantile detection plots of extreme outliers on all data sets.

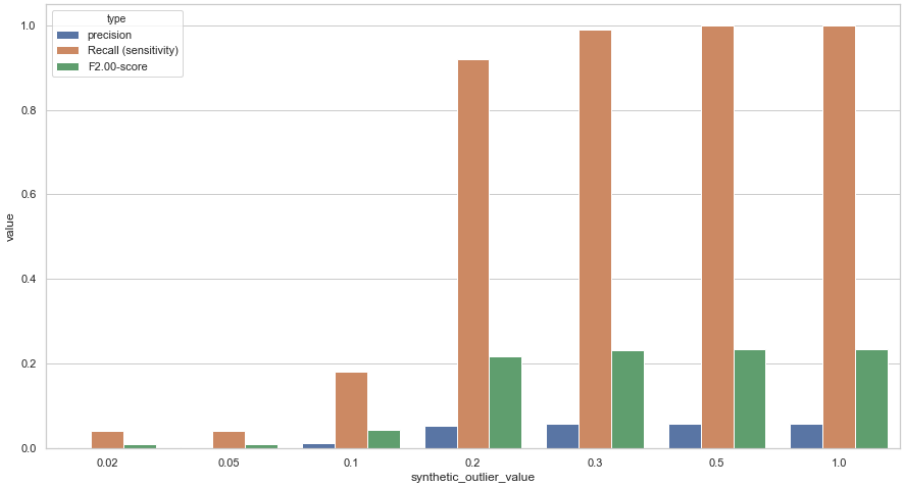
Figure A.4 shows how well different outliers types are detected for different outlier generation settings. The bigger the synthetic outlier value, the more easily the outlier is detected. The range of depicted values is bigger than in figures of the main sections, which focused on the most important values.



(a) Drifts.



(b) Jumps.



(c) Extremes.

Figure A.4: 104OYE barplots for all outliers values.

Appendix B

Synthetic evaluation results

In the following sections, the results of the most interesting outlier parameters of the synthetic evaluation results are shown. For brevity, we abbreviated the columns in the following way: C: outlier category, V: outlier value, TP: true positive, FP: false positive, TN: true negative, FN: false negative, Acc: accuracy, Prec: precision, Rec: Recall, OP: Optimised precision, F2: F2.00-score, PTM: proportion totally missed, DNT: drift detected in n time steps.

In the outlier category column, we denote drift as D, extremes as E and jumps as J.

To keep the results concise, we only report the data for outlier values 0.05, 0.1 and 0.2. These outlier values are the most interesting ones. This means that we do not show the results of values 0.02, 0.3, 0.5 and 1.0 here.

B.1 Multivariate model results

B.1.1 Tables multi layer perceptron

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	10289	3004	8840	7231	0.651	0.769	0.587	0.478	0.613	0	683.18
D	0.1	12362	3009	8835	5158	0.722	0.813	0.706	0.609	0.722	0	669.63
D	0.2	14278	3024	8820	3242	0.787	0.832	0.815	0.656	0.814	0	512.96
E	0.05	68	11351	17913	32	0.612	0.006	0.68	0.56	0.029		
E	0.1	98	11351	17913	2	0.613	0.009	0.98	0.382	0.041		
E	0.2	100	11541	17723	0	0.607	0.009	1	0.361	0.042		
J	0.05	2516	9424	15620	1804	0.618	0.2	0.582	0.295	0.419	0	
J	0.1	3742	9432	15612	578	0.659	0.278	0.866	0.409	0.606	0	
J	0.2	4312	9444	15600	8	0.678	0.316	0.998	0.445	0.696	0	

(a) 102BFS

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	10289	3004	8840	7231	0.651	0.769	0.587	0.478	0.613	0	683.18
D	0.1	12362	3009	8835	5158	0.722	0.813	0.706	0.609	0.722	0	669.63
D	0.2	14278	3024	8820	3242	0.787	0.832	0.815	0.656	0.814	0	512.96
E	0.05	68	11351	17913	32	0.612	0.006	0.68	0.56	0.029		
E	0.1	98	11351	17913	2	0.613	0.009	0.98	0.382	0.041		
E	0.2	100	11541	17723	0	0.607	0.009	1	0.361	0.042		
J	0.05	2516	9424	15620	1804	0.618	0.2	0.582	0.295	0.419	0	
J	0.1	3742	9432	15612	578	0.659	0.278	0.866	0.409	0.606	0	
J	0.2	4312	9444	15600	8	0.678	0.316	0.998	0.445	0.696	0	

(b) 103HOE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	3430	607	11237	14090	0.499	0.658	0.196	-0.199	0.223	0	2981.19
D	0.1	6992	614	11230	10528	0.621	0.882	0.399	0.162	0.435	0	3607.11
D	0.2	12226	626	11218	5294	0.798	0.951	0.698	0.639	0.733	0	2724.87
E	0.05	4	1628	27636	96	0.941	0.002	0.04	0.023	0.01		
E	0.1	18	1628	27636	82	0.942	0.011	0.18	0.262	0.044		
E	0.2	92	1628	27636	8	0.944	0.053	0.92	0.931	0.217		
J	0.05	2008	1380	23664	2312	0.874	0.441	0.465	0.41	0.453	0.13	
J	0.1	3749	1392	23652	571	0.933	0.725	0.868	0.856	0.832	0	
J	0.2	4292	1416	23628	28	0.951	0.753	0.993	0.924	0.933	0	

(c) 104OYE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	2660	382	11462	14860	0.481	0.803	0.152	-0.268	0.178	0	1554.04
D	0.1	4341	386	11458	13179	0.538	0.753	0.248	-0.112	0.277	0.02	1810.78
D	0.2	7212	390	11454	10308	0.636	0.837	0.412	0.127	0.433	0.01	3026.98
E	0.05	8	2585	26679	92	0.909	0.003	0.08	0.07	0.013		
E	0.1	8	2585	26679	92	0.909	0.003	0.08	0.07	0.013		
E	0.2	9	2490	26774	91	0.912	0.004	0.09	0.091	0.016		
J	0.05	1111	2089	22955	3209	0.82	0.238	0.257	0.14	0.25	0.38	
J	0.1	1886	2093	22951	2434	0.846	0.321	0.437	0.304	0.405	0.48	
J	0.2	2797	2103	22941	1523	0.877	0.489	0.647	0.564	0.602	0.18	

(d) 201D

Table B.1: Multivariate MLP results.

B.1.2 Tables perceptron model

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	1814	487	11357	15706	0.449	0.711	0.104	-0.369	0.125	0.06	4293.82
D	0.1	5875	490	11354	11645	0.587	0.92	0.335	0.096	0.382	0	3021.88
D	0.2	11118	505	11339	6402	0.765	0.961	0.635	0.557	0.678	0	2918.89
E	0.05	14	1447	27817	86	0.948	0.01	0.14	0.205	0.038		
E	0.1	49	1447	27817	51	0.949	0.033	0.49	0.629	0.129		
E	0.2	100	1447	27817	0	0.951	0.065	1	0.925	0.257		
J	0.05	1102	1208	23836	3218	0.849	0.33	0.255	0.191	0.264	0.24	
J	0.1	2885	1216	23828	1435	0.91	0.686	0.668	0.677	0.657	0	
J	0.2	4291	1237	23807	29	0.957	0.785	0.993	0.929	0.941	0	

(a) 102BFS

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	2844	312	11532	14676	0.490	0.901	0.162	-0.236	0.191	0.000	3770.950
D	0.1	3978	318	11526	13542	0.528	0.904	0.227	-0.114	0.263	0.000	3580.840
D	0.2	8284	323	11521	9236	0.674	0.952	0.473	0.307	0.520	0.000	3294.180
E	0.05	16	1963	27301	84	0.930	0.008	0.160	0.223	0.034		
E	0.1	27	1963	27301	73	0.931	0.014	0.270	0.380	0.056		
E	0.2	79	1963	27301	21	0.932	0.039	0.790	0.849	0.162		
J	0.05	927	1633	23411	3393	0.829	0.286	0.215	0.125	0.219	0.230	
J	0.1	1819	1639	23405	2501	0.859	0.433	0.421	0.381	0.415	0.090	
J	0.2	4049	1654	23390	271	0.934	0.715	0.937	0.890	0.879	0.000	

(b) 103HOE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	7778	1283	10561	9742	0.625	0.734	0.444	0.186	0.463	0.000	617.670
D	0.1	9914	1287	10557	7606	0.697	0.832	0.566	0.375	0.584	0.000	911.970
D	0.2	13630	1295	10549	3890	0.823	0.909	0.778	0.706	0.796	0.000	1132.780
E	0.05	21	7374	21890	79	0.746	0.003	0.210	0.185	0.013		
E	0.1	55	7469	21795	45	0.744	0.007	0.550	0.594	0.035		
E	0.2	93	7564	21700	7	0.742	0.012	0.930	0.629	0.058		
J	0.05	2423	6159	18885	1897	0.726	0.251	0.561	0.315	0.447	0.120	
J	0.1	3231	6167	18877	1089	0.753	0.332	0.748	0.541	0.596	0.000	
J	0.2	4277	6182	18862	43	0.788	0.411	0.990	0.652	0.771	0.000	

(c) 104OYE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	8768	6664	5180	8752	0.475	0.567	0.500	0.353	0.512	0.000	14.370
D	0.1	11142	6670	5174	6378	0.556	0.627	0.636	0.296	0.631	0.000	14.160
D	0.2	13483	6678	5166	4037	0.635	0.673	0.770	0.298	0.743	0.000	14.150
E	0.05	51	13821	15443	49	0.528	0.004	0.510	0.511	0.018		
E	0.1	67	13821	15443	33	0.528	0.005	0.670	0.409	0.023		
E	0.2	92	13821	15443	8	0.529	0.007	0.920	0.258	0.032		
J	0.05	2413	11955	13089	1907	0.528	0.165	0.559	0.240	0.377	0.030	
J	0.1	2958	11966	13078	1362	0.546	0.193	0.685	0.274	0.452	0.010	
J	0.2	4236	11978	13066	84	0.589	0.264	0.980	0.282	0.634	0.000	

(d) 201D

Table B.2: Multivariate perceptron results.

B.1.3 Tables RNNs

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	6509	1709	10117	11011	0.567	0.770	0.371	0.137	0.409	0.000	811.900
D	0.1	9243	1716	10110	8277	0.659	0.843	0.528	0.382	0.562	0.000	807.680
D	0.2	12581	1726	10099	4939	0.773	0.896	0.718	0.645	0.740	0.000	1397.240
E	0.05	28	5487	23759	71	0.811	0.005	0.283	0.327	0.024		
E	0.1	64	5487	23759	35	0.812	0.012	0.646	0.698	0.054		
E	0.2	96	5392	23854	3	0.816	0.017	0.970	0.730	0.082		
J	0.05	1748	4564	20461	2571	0.757	0.243	0.405	0.287	0.354	0.180	
J	0.1	3134	4571	20454	1186	0.804	0.377	0.725	0.597	0.609	0.060	
J	0.2	4236	4587	20439	84	0.841	0.488	0.981	0.741	0.812	0.000	

(a) 102BFS

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	3189	811	11014	14331	0.484	0.754	0.182	-0.206	0.212	0.000	3130.130
D	0.1	5022	817	11009	12497	0.546	0.813	0.287	-0.008	0.325	0.000	2976.870
D	0.2	9317	822	11003	8203	0.692	0.905	0.532	0.404	0.576	0.000	2583.420
E	0.05	9	2781	26465	90	0.902	0.003	0.091	0.085	0.014		
E	0.1	25	2781	26465	74	0.903	0.009	0.253	0.339	0.039		
E	0.2	59	2781	26465	40	0.904	0.021	0.596	0.698	0.091		
J	0.05	1280	2337	22688	3040	0.817	0.289	0.296	0.222	0.291	0.230	
J	0.1	2250	2343	22682	2070	0.850	0.415	0.521	0.467	0.490	0.070	
J	0.2	4061	2358	22667	259	0.911	0.635	0.940	0.865	0.856	0.000	

(b) 103HOE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	601	14	11812	16918	0.423	0.963	0.034	-0.511	0.042	0.000	2819.050
D	0.1	3195	15	11810	14325	0.511	0.994	0.182	-0.186	0.216	0.000	3885.680
D	0.2	9528	26	11799	7991	0.727	0.997	0.544	0.429	0.597	0.000	3413.280
E	0.05	1	310	28936	98	0.986	0.003	0.010	0.006	0.007		
E	0.1	11	310	28936	88	0.986	0.034	0.111	0.188	0.077		
E	0.2	52	310	28936	47	0.988	0.144	0.525	0.681	0.343		
J	0.05	624	261	24764	3696	0.865	0.482	0.144	0.092	0.166	0.280	
J	0.1	2239	268	24757	2081	0.920	0.829	0.518	0.552	0.547	0.030	
J	0.2	4275	290	24736	45	0.989	0.937	0.990	0.981	0.978	0.000	

(c) 104OYE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	0	0	11825	17520	0.403	0.000	0.000	-0.597	0.000	1.000	17519.680
D	0.1	2	0	11825	17518	0.403	0.350	0.000	-0.597	0.000	0.650	16093.800
D	0.2	76	0	11825	17443	0.406	0.500	0.004	-0.586	0.005	0.500	13261.840
E	0.05	0	0	29246	99	0.997	0.000	0.000	-0.003	0.000		
E	0.1	0	0	29246	99	0.997	0.000	0.000	-0.003	0.000		
E	0.2	1	0	29246	98	0.997	1.000	0.010	0.017	0.013		
J	0.05	0	0	25025	4319	0.853	0.080	0.000	-0.147	0.000	0.920	
J	0.1	6	0	25025	4314	0.853	0.180	0.001	-0.144	0.002	0.820	
J	0.2	91	0	25025	4229	0.856	0.360	0.021	-0.107	0.026	0.640	

(d) 201D

Table B.3: Multivariate RNN results.

B.1.4 Tables quantile regression forests

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	4758	1143	10701	12762	0.526	0.804	0.272	-0.013	0.313	0.000	1035.450
D	0.1	8121	1151	10693	9399	0.641	0.874	0.464	0.315	0.510	0.000	1443.970
D	0.2	11897	1161	10683	5623	0.769	0.910	0.679	0.624	0.714	0.000	1672.830
E	0.05	27	4097	25167	73	0.858	0.007	0.270	0.336	0.030		
E	0.1	58	4002	25262	42	0.862	0.014	0.580	0.666	0.065		
E	0.2	100	4002	25262	0	0.864	0.024	1.000	0.790	0.111		
J	0.05	1699	3285	21759	2621	0.799	0.309	0.393	0.359	0.371	0.100	
J	0.1	3344	3297	21747	976	0.854	0.494	0.774	0.720	0.692	0.000	
J	0.2	4319	3319	21725	1	0.887	0.568	1.000	0.816	0.867	0.000	

(a) 102BFS

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	3908	5651	6193	13612	0.344	0.393	0.223	-0.068	0.243	0.000	1274.930
D	0.1	4916	5652	6192	12604	0.378	0.444	0.281	0.063	0.301	0.000	1270.470
D	0.2	9955	5653	6191	7565	0.550	0.642	0.568	0.395	0.579	0.000	1248.640
E	0.05	38	10043	19221	62	0.656	0.004	0.380	0.389	0.018		
E	0.1	60	10138	19126	40	0.653	0.006	0.600	0.611	0.028		
E	0.2	94	10138	19126	6	0.655	0.009	0.940	0.475	0.044		
J	0.05	1414	8646	16398	2906	0.607	0.132	0.327	0.135	0.252	0.040	
J	0.1	2781	8650	16394	1539	0.653	0.232	0.644	0.389	0.472	0.000	
J	0.2	4068	8667	16378	252	0.696	0.323	0.942	0.517	0.678	0.000	

(b) 103HOE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	6209	1824	10020	11311	0.553	0.675	0.354	0.084	0.384	0.000	421.800
D	0.1	8426	1829	10015	9094	0.628	0.763	0.481	0.291	0.507	0.000	398.180
D	0.2	11332	1839	10005	6188	0.727	0.844	0.647	0.536	0.669	0.000	595.230
E	0.05	39	6904	22360	61	0.763	0.006	0.390	0.439	0.027		
E	0.1	53	6809	22455	47	0.767	0.008	0.530	0.584	0.036		
E	0.2	83	6975	22289	17	0.762	0.012	0.830	0.719	0.056		
J	0.05	2027	5873	19171	2293	0.722	0.230	0.469	0.342	0.387	0.070	
J	0.1	3035	5880	19164	1285	0.756	0.320	0.703	0.535	0.565	0.020	
J	0.2	4248	5892	19152	72	0.797	0.421	0.983	0.672	0.775	0.000	

(c) 104OYE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	167	0	11844	17353	0.409	0.510	0.010	-0.573	0.012	0.490	13148.940
D	0.1	3354	4	11840	14166	0.517	0.729	0.191	-0.207	0.218	0.270	9842.860
D	0.2	4625	9	11835	12895	0.561	0.909	0.264	-0.101	0.289	0.090	8824.700
E	0.05	0	0	29264	100	0.997	0.000	0.000	-0.003	0.000		
E	0.1	0	0	29264	100	0.997	0.000	0.000	-0.003	0.000		
E	0.2	17	0	29264	83	0.997	1.000	0.170	0.288	0.204		
J	0.05	337	0	25044	3983	0.864	0.320	0.078	-0.023	0.089	0.680	
J	0.1	1039	5	25039	3281	0.888	0.487	0.241	0.155	0.249	0.510	
J	0.2	2071	15	25029	2249	0.923	0.896	0.480	0.437	0.490	0.100	

(d) 201D

Table B.4: Multivariate QRF results

B.1.5 Tables linear regression

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	5476	909	10935	12044	0.559	0.854	0.313	0.060	0.357	0.000	1450.790
D	0.1	9890	918	10926	7630	0.709	0.917	0.564	0.465	0.610	0.000	1355.180
D	0.2	12905	930	10914	4615	0.811	0.936	0.737	0.695	0.768	0.000	1192.910
E	0.05	35	5250	24014	65	0.819	0.007	0.350	0.417	0.031		
E	0.1	73	5250	24014	27	0.820	0.014	0.730	0.762	0.064		
E	0.2	100	5250	24014	0	0.821	0.019	1.000	0.723	0.087		
J	0.05	1991	4408	20636	2329	0.771	0.295	0.461	0.422	0.411	0.000	
J	0.1	3811	4419	20625	509	0.832	0.461	0.882	0.707	0.740	0.000	
J	0.2	4320	4441	20603	0	0.849	0.500	1.000	0.751	0.831	0.000	

(a) 102BFS

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	6574	3856	7988	10946	0.496	0.608	0.375	0.186	0.403	0.000	1490.860
D	0.1	8443	3856	7988	9077	0.560	0.665	0.482	0.340	0.506	0.000	1763.920
D	0.2	11713	3862	7982	5807	0.671	0.741	0.669	0.548	0.678	0.000	1636.290
E	0.05	38	8617	20647	62	0.704	0.004	0.380	0.405	0.021		
E	0.1	60	8617	20647	40	0.705	0.007	0.600	0.624	0.033		
E	0.2	100	8807	20457	0	0.700	0.011	1.000	0.523	0.054		
J	0.05	1932	7426	17618	2388	0.666	0.189	0.447	0.266	0.350	0.060	
J	0.1	2853	7433	17611	1467	0.697	0.258	0.661	0.411	0.502	0.030	
J	0.2	4159	7445	17599	161	0.741	0.359	0.963	0.586	0.720	0.000	

(b) 103HOE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	0	0	11844	17520	0.403	0.000	0.000	-0.597	0.000	1.000	17520.000
D	0.1	92	0	11844	17428	0.406	0.380	0.005	-0.583	0.007	0.620	15856.240
D	0.2	2741	3	11841	14779	0.497	0.978	0.156	-0.246	0.185	0.020	10462.490
E	0.05	0	0	29264	100	0.997	0.000	0.000	-0.003	0.000		
E	0.1	1	0	29264	99	0.997	1.000	0.010	0.016	0.012		
E	0.2	9	0	29264	91	0.997	1.000	0.090	0.162	0.110		
J	0.05	1	0	25044	4319	0.853	0.080	0.000	-0.147	0.000	0.920	
J	0.1	185	0	25044	4135	0.859	0.429	0.043	-0.066	0.052	0.570	
J	0.2	3078	5	25039	1242	0.958	0.969	0.713	0.737	0.735	0.030	

(c) 104OYE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	11	7	11837	17509	0.403	0.592	0.001	-0.595	0.001	0.130	5511.140
D	0.1	95	7	11837	17425	0.406	0.729	0.005	-0.583	0.007	0.130	5576.590
D	0.2	4554	10	11834	12966	0.558	0.997	0.260	-0.037	0.303	0.000	4551.080
E	0.05	0	16	29248	100	0.996	0.000	0.000	-0.004	0.000		
E	0.1	0	16	29248	100	0.996	0.000	0.000	-0.004	0.000		
E	0.2	25	16	29248	75	0.997	0.610	0.250	0.397	0.283		
J	0.05	10	13	25031	4310	0.853	0.205	0.002	-0.142	0.003	0.600	
J	0.1	433	14	25030	3887	0.867	0.609	0.100	0.012	0.113	0.300	
J	0.2	3457	24	25020	863	0.970	0.992	0.800	0.833	0.821	0.000	

(d) 201D

Table B.5: Multivariate linear regression results.

B.2 Univariate model results

B.2.1 Tables multi layer perceptron

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	829	733	10285	16691	0.389	0.529	0.047	-0.514	0.058	0.000	1316.930
D	0.1	787	926	10092	16733	0.381	0.460	0.045	-0.525	0.055	0.000	1354.370
D	0.2	837	1423	9595	16683	0.366	0.383	0.048	-0.529	0.057	0.000	1443.650
E	0.05	8	1531	26910	89	0.943	0.005	0.082	0.104	0.021		
E	0.1	78	1531	26910	19	0.946	0.048	0.804	0.865	0.195		
E	0.2	96	1722	26719	1	0.940	0.053	0.990	0.914	0.218		
J	0.05	245	1472	22746	4075	0.806	0.140	0.057	-0.084	0.064	0.150	
J	0.1	328	1781	22437	3992	0.798	0.154	0.076	-0.057	0.084	0.010	
J	0.2	751	2305	21913	3569	0.794	0.258	0.174	0.089	0.184	0.000	

(a) 102BFS

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	711	182	10836	16809	0.405	0.796	0.041	-0.516	0.050	0.000	2103.170
D	0.1	641	214	10804	16879	0.401	0.756	0.037	-0.527	0.045	0.000	2103.170
D	0.2	522	217	10801	16998	0.397	0.709	0.030	-0.544	0.037	0.000	2123.330
E	0.05	2	862	27579	95	0.966	0.002	0.021	0.008	0.008		
E	0.1	6	862	27579	91	0.967	0.007	0.062	0.087	0.024		
E	0.2	74	862	27579	23	0.969	0.079	0.763	0.850	0.279		
J	0.05	141	717	23501	4179	0.828	0.160	0.033	-0.108	0.039	0.190	
J	0.1	194	736	23482	4126	0.830	0.200	0.045	-0.084	0.053	0.100	
J	0.2	187	733	23485	4133	0.829	0.198	0.043	-0.086	0.051	0.020	

(b) 103HOE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	405	150	10868	17115	0.395	0.724	0.023	-0.560	0.029	0.000	2916.670
D	0.1	447	198	10820	17073	0.395	0.704	0.026	-0.555	0.032	0.000	2834.920
D	0.2	491	283	10735	17029	0.393	0.634	0.028	-0.551	0.035	0.000	2824.540
E	0.05	4	506	27935	93	0.979	0.008	0.041	0.060	0.022		
E	0.1	20	506	27935	77	0.980	0.038	0.206	0.327	0.109		
E	0.2	87	509	27932	10	0.982	0.146	0.897	0.936	0.442		
J	0.05	117	427	23791	4203	0.838	0.206	0.027	-0.110	0.033	0.210	
J	0.1	185	484	23734	4135	0.838	0.261	0.043	-0.080	0.051	0.090	
J	0.2	285	589	23629	4035	0.838	0.322	0.066	-0.037	0.078	0.000	

(c) 104OYE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	45	51	10967	17475	0.386	0.470	0.003	-0.609	0.003	0.530	10489.270
D	0.1	46	72	10946	17474	0.385	0.415	0.003	-0.610	0.003	0.500	10489.090
D	0.2	60	109	10909	17460	0.384	0.385	0.003	-0.609	0.004	0.310	10393.060
E	0.05	0	96	28345	97	0.993	0.000	0.000	-0.007	0.000		
E	0.1	0	96	28345	97	0.993	0.000	0.000	-0.007	0.000		
E	0.2	0	96	28345	97	0.993	0.000	0.000	-0.007	0.000		
J	0.05	15	77	24141	4305	0.846	0.160	0.004	-0.147	0.004	0.840	
J	0.1	12	95	24123	4308	0.846	0.110	0.003	-0.149	0.004	0.810	
J	0.2	89	141	24077	4231	0.847	0.400	0.021	-0.113	0.025	0.130	

(d) 201D

Table B.6: Univariate MLP results.

B.2.2 Tables perceptron model

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	1564	806	10212	15956	0.413	0.654	0.089	-0.418	0.108	0.000	1181.510
D	0.1	2882	898	10120	14638	0.456	0.695	0.164	-0.263	0.192	0.000	1267.910
D	0.2	5914	1074	9944	11606	0.556	0.688	0.338	0.013	0.364	0.000	1276.550
E	0.05	11	2295	26146	86	0.917	0.005	0.113	0.136	0.020		
E	0.1	66	2295	26146	31	0.918	0.028	0.680	0.769	0.120		
E	0.2	97	2295	26146	0	0.920	0.041	1.000	0.878	0.174		
J	0.05	680	1880	22338	3640	0.807	0.226	0.157	0.053	0.166	0.130	
J	0.1	1438	1989	22229	2882	0.829	0.334	0.333	0.273	0.328	0.010	
J	0.2	2235	2172	22046	2085	0.851	0.382	0.517	0.385	0.476	0.000	

(a) 102BFS

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	1118	426	10592	16402	0.410	0.734	0.064	-0.466	0.078	0.000	2095.450
D	0.1	1859	669	10349	15661	0.428	0.710	0.106	-0.381	0.126	0.000	2106.010
D	0.2	2444	1268	9750	15076	0.427	0.613	0.140	-0.328	0.163	0.000	2125.900
E	0.05	12	1245	27196	85	0.953	0.010	0.124	0.183	0.036		
E	0.1	17	1150	27291	80	0.957	0.015	0.175	0.266	0.055		
E	0.2	87	1245	27196	10	0.956	0.065	0.897	0.924	0.253		
J	0.05	423	1113	23105	3897	0.824	0.235	0.098	-0.009	0.109	0.100	
J	0.1	812	1283	22935	3508	0.832	0.306	0.188	0.101	0.197	0.050	
J	0.2	1688	1787	22431	2632	0.845	0.428	0.391	0.330	0.390	0.000	

(b) 103HOE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	973	296	10722	16547	0.410	0.769	0.056	-0.483	0.068	0.000	2608.140
D	0.1	929	400	10618	16591	0.405	0.703	0.053	-0.491	0.065	0.000	2608.170
D	0.2	945	726	10292	16575	0.394	0.585	0.054	-0.497	0.066	0.000	2608.190
E	0.05	6	1261	27180	91	0.953	0.005	0.062	0.074	0.018		
E	0.1	14	1261	27180	83	0.953	0.011	0.144	0.215	0.042		
E	0.2	86	1261	27180	11	0.955	0.064	0.887	0.918	0.248		
J	0.05	207	1033	23185	4113	0.820	0.165	0.048	-0.089	0.056	0.210	
J	0.1	287	1122	23096	4033	0.819	0.202	0.066	-0.054	0.077	0.050	
J	0.2	430	1419	22799	3890	0.814	0.234	0.100	0.001	0.112	0.000	

(c) 104OYE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	6695	3934	7084	10825	0.483	0.628	0.382	0.223	0.414	0.000	72.090
D	0.1	8773	4056	6962	8747	0.551	0.646	0.501	0.284	0.515	0.000	108.710
D	0.2	9454	4439	6579	8066	0.562	0.578	0.540	0.159	0.533	0.000	133.670
E	0.05	35	10557	17884	62	0.628	0.003	0.361	0.357	0.016		
E	0.1	35	10557	17884	62	0.628	0.003	0.361	0.357	0.016		
E	0.2	57	10366	18075	40	0.635	0.005	0.588	0.596	0.026		
J	0.05	1727	8909	15309	2593	0.597	0.157	0.400	0.132	0.303	0.190	
J	0.1	2308	8968	15250	2012	0.615	0.186	0.534	0.103	0.385	0.010	
J	0.2	2292	9254	14964	2028	0.605	0.180	0.531	0.077	0.378	0.000	

(d) 201D

Table B.7: Univariate perceptron results.

B.2.3 Tables linear regression

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	450	155	10863	17070	0.396	0.743	0.026	-0.553	0.032	0.000	2187.800
D	0.1	430	246	10772	17090	0.393	0.634	0.025	-0.559	0.030	0.000	2180.980
D	0.2	437	335	10683	17083	0.390	0.561	0.025	-0.560	0.031	0.000	2180.360
E	0.05	3	632	27806	97	0.974	0.005	0.030	0.034	0.014		
E	0.1	90	632	27806	10	0.978	0.125	0.900	0.936	0.401		
E	0.2	100	631	27807	0	0.978	0.137	1.000	0.967	0.442		
J	0.05	104	475	23743	4216	0.836	0.176	0.024	-0.117	0.029	0.300	
J	0.1	218	581	23637	4102	0.836	0.271	0.051	-0.067	0.060	0.010	
J	0.2	299	675	23543	4021	0.835	0.304	0.069	-0.033	0.082	0.000	

(a) 102BFS

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	560	174	10844	16960	0.400	0.761	0.032	-0.538	0.040	0.000	1706.270
D	0.1	572	204	10814	16948	0.399	0.737	0.033	-0.537	0.040	0.000	1706.270
D	0.2	620	325	10693	16900	0.396	0.654	0.035	-0.533	0.044	0.000	1706.850
E	0.05	5	767	27671	95	0.970	0.006	0.050	0.068	0.021		
E	0.1	31	766	27672	69	0.971	0.039	0.310	0.454	0.129		
E	0.2	100	766	27672	0	0.973	0.115	1.000	0.960	0.395		
J	0.05	126	635	23583	4194	0.831	0.165	0.029	-0.113	0.035	0.150	
J	0.1	162	665	23553	4158	0.831	0.191	0.038	-0.097	0.045	0.080	
J	0.2	323	797	23421	3997	0.832	0.285	0.075	-0.026	0.088	0.000	

(b) 103HOE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	0	0	11018	17520	0.386	0.000	0.000	-0.614	0.000	1.000	17520.000
D	0.1	0	0	11018	17520	0.386	0.000	0.000	-0.614	0.000	1.000	17520.000
D	0.2	0	4	11014	17520	0.386	0.000	0.000	-0.614	0.000	0.990	17519.990
E	0.05	0	0	28438	100	0.996	0.000	0.000	-0.004	0.000		
E	0.1	0	0	28438	100	0.996	0.000	0.000	-0.004	0.000		
E	0.2	2	0	28438	98	0.997	1.000	0.020	0.036	0.025		
J	0.05	0	0	24218	4320	0.849	0.010	0.000	-0.151	0.000	0.990	
J	0.1	1	0	24218	4319	0.849	0.020	0.000	-0.151	0.000	0.980	
J	0.2	6	6	24212	4314	0.849	0.048	0.001	-0.149	0.002	0.930	

(c) 104OYE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	7	7	11011	17513	0.386	0.470	0.000	-0.613	0.000	0.530	10509.480
D	0.1	7	8	11010	17513	0.386	0.468	0.000	-0.613	0.000	0.530	10509.480
D	0.2	8	19	10999	17512	0.386	0.476	0.000	-0.613	0.001	0.300	7943.210
E	0.05	0	14	28424	100	0.996	0.000	0.000	-0.004	0.000		
E	0.1	0	14	28424	100	0.996	0.000	0.000	-0.004	0.000		
E	0.2	6	14	28424	94	0.996	0.300	0.060	0.109	0.071		
J	0.05	3	12	24207	4317	0.848	0.182	0.001	-0.150	0.001	0.790	
J	0.1	5	11	24207	4315	0.848	0.195	0.001	-0.150	0.001	0.720	
J	0.2	29	27	24191	4291	0.849	0.347	0.007	-0.138	0.008	0.490	

(d) 201D

Table B.8: Univariate linear regression results.

B.2.4 Tables auto-regression

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	3	5	11013	17517	0.386	0.319	0.000	-0.614	0.000	0.000	3545.520
D	0.1	3	5	11013	17517	0.386	0.319	0.000	-0.614	0.000	0.000	3545.520
D	0.2	3	5	11013	17517	0.386	0.319	0.000	-0.614	0.000	0.000	3545.520
E	0.05	96	102	28339	1	0.996	0.485	0.990	0.993	0.819		
E	0.1	97	104	28337	0	0.996	0.483	1.000	0.995	0.823		
E	0.2	97	104	28337	0	0.996	0.483	1.000	0.995	0.823		
J	0.05	2	7	24211	4318	0.848	0.191	0.000	-0.151	0.000	0.000	
J	0.1	2	7	24211	4318	0.848	0.190	0.000	-0.151	0.000	0.010	
J	0.2	2	7	24211	4318	0.848	0.191	0.000	-0.151	0.000	0.000	

(a) 102BFS

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	8	7	11011	17512	0.386	0.532	0.000	-0.613	0.001	0.000	4138.150
D	0.1	8	7	11011	17512	0.386	0.538	0.000	-0.613	0.001	0.000	4138.150
D	0.2	8	7	11011	17512	0.386	0.544	0.000	-0.613	0.001	0.000	3617.050
E	0.05	96	111	28330	1	0.996	0.464	0.990	0.993	0.807		
E	0.1	97	111	28330	0	0.996	0.466	1.000	0.994	0.814		
E	0.2	97	112	28329	0	0.996	0.464	1.000	0.994	0.812		
J	0.05	3	13	24205	4317	0.848	0.193	0.001	-0.150	0.001	0.000	
J	0.1	3	13	24205	4317	0.848	0.193	0.001	-0.150	0.001	0.000	
J	0.2	3	13	24205	4317	0.848	0.193	0.001	-0.150	0.001	0.000	

(b) 103HOE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	1	12	11006	17519	0.386	0.086	0.000	-0.614	0.000	0.520	9646.190
D	0.1	1	12	11006	17519	0.386	0.086	0.000	-0.614	0.000	0.520	9646.190
D	0.2	1	13	11005	17519	0.386	0.082	0.000	-0.614	0.000	0.520	9646.190
E	0.05	97	109	28332	0	0.996	0.471	1.000	0.994	0.816		
E	0.1	97	109	28332	0	0.996	0.471	1.000	0.994	0.816		
E	0.2	97	109	28332	0	0.996	0.471	1.000	0.994	0.816		
J	0.05	1	12	24206	4319	0.848	0.107	0.000	-0.151	0.000	0.010	
J	0.1	2	12	24206	4319	0.848	0.108	0.000	-0.151	0.000	0.000	
J	0.2	2	13	24205	4318	0.848	0.139	0.000	-0.151	0.001	0.000	

(c) 104OYE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	8	11	11041	17478	0.387	0.416	0.000	-0.612	0.001	0.000	2167.240
D	0.1	8	11	11041	17478	0.387	0.414	0.000	-0.612	0.001	0.000	2167.240
D	0.2	8	12	11040	17478	0.387	0.394	0.000	-0.612	0.001	0.000	2167.240
E	0.05	95	115	28326	2	0.996	0.452	0.979	0.988	0.794		
E	0.1	97	117	28324	0	0.996	0.453	1.000	0.994	0.806		
E	0.2	97	212	28229	0	0.993	0.314	1.000	0.989	0.696		
J	0.05	2	17	24201	4318	0.848	0.123	0.001	-0.151	0.001	0.010	
J	0.1	3	18	24200	4317	0.848	0.126	0.001	-0.151	0.001	0.000	
J	0.2	3	19	24199	4317	0.848	0.158	0.001	-0.150	0.001	0.000	

(d) 201D

Table B.9: Univariate AR results.

B.2.5 Tables isolation forest

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	6775	1723	9295	10745	0.563	0.806	0.387	0.173	0.426	0.000	1037.160
D	0.1	9569	2363	8655	7951	0.639	0.790	0.546	0.358	0.571	0.000	1143.280
D	0.2	13310	3137	7881	4210	0.743	0.811	0.760	0.599	0.764	0.000	1632.750
E	0.05	25	7503	20938	72	0.735	0.003	0.258	0.253	0.016		
E	0.1	26	7525	20916	71	0.734	0.003	0.268	0.268	0.016		
E	0.2	26	7539	20902	71	0.733	0.003	0.268	0.268	0.016		
J	0.05	1939	6301	17917	2381	0.696	0.232	0.449	0.270	0.369	0.250	
J	0.1	2906	6797	17421	1414	0.712	0.291	0.673	0.459	0.523	0.040	
J	0.2	4191	7495	16723	129	0.733	0.372	0.970	0.558	0.726	0.000	

(a) 102BFS

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	4153	5398	5620	13367	0.342	0.434	0.237	-0.026	0.260	0.000	2362.390
D	0.1	5790	5437	5581	11730	0.398	0.488	0.330	0.030	0.347	0.000	2328.220
D	0.2	9011	5712	5306	8509	0.502	0.604	0.514	0.285	0.524	0.000	2082.370
E	0.05	35	10217	18224	62	0.640	0.003	0.361	0.360	0.016		
E	0.1	35	10220	18221	62	0.640	0.003	0.361	0.360	0.016		
E	0.2	35	10219	18222	62	0.640	0.003	0.361	0.360	0.016		
J	0.05	1488	8892	15326	2832	0.589	0.133	0.344	0.088	0.260	0.190	
J	0.1	2246	8992	15226	2074	0.612	0.180	0.520	0.182	0.375	0.100	
J	0.2	3644	9375	14843	676	0.648	0.273	0.843	0.398	0.592	0.010	

(b) 103HOE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	5148	3023	7995	12372	0.461	0.607	0.294	0.018	0.326	0.000	3481.990
D	0.1	6584	3129	7889	10936	0.507	0.649	0.376	0.167	0.407	0.000	2440.650
D	0.2	9394	3412	7606	8126	0.596	0.709	0.536	0.409	0.559	0.000	2141.660
E	0.05	20	6017	22424	77	0.786	0.003	0.206	0.201	0.016		
E	0.1	20	6037	22404	77	0.786	0.003	0.206	0.201	0.016		
E	0.2	20	6060	22381	77	0.785	0.003	0.206	0.200	0.015		
J	0.05	2039	5178	19040	2281	0.739	0.256	0.472	0.319	0.401	0.170	
J	0.1	2215	5293	18925	2105	0.741	0.276	0.513	0.396	0.434	0.060	
J	0.2	2468	5652	18566	1852	0.737	0.291	0.571	0.452	0.476	0.000	

(c) 104OYE

C	V	TP	FP	TN	FN	Acc	Prec	Rec	OP	F2	PTM	DNT
D	0.05	0	0	11018	17520	0.386	0.000	0.000	-0.614	0.000	1.000	17520.000
D	0.1	0	1	11017	17520	0.386	0.000	0.000	-0.614	0.000	1.000	17520.000
D	0.2	1086	992	10026	16434	0.389	0.265	0.062	-0.480	0.073	0.500	13802.180
E	0.05	0	0	28441	97	0.997	0.000	0.000	-0.003	0.000		
E	0.1	0	0	28441	97	0.997	0.000	0.000	-0.003	0.000		
E	0.2	0	0	28441	97	0.997	0.000	0.000	-0.003	0.000		
J	0.05	0	0	24218	4320	0.849	0.000	0.000	-0.151	0.000	1.000	
J	0.1	173	61	24157	4147	0.853	0.236	0.040	-0.078	0.047	0.690	
J	0.2	818	755	23463	3502	0.851	0.291	0.189	0.132	0.199	0.500	

(d) 201D

Table B.10: Univariate IF results.

Appendix C

Hyperband tuner results

In this section, we show the best 10 runs of the Hyperband tuner per used sensor.

C.1 Quantile regression: MLP in multivariate setting

μ(loss)	d	lr	l	units
0.1750	0.4	0.0005	1	128
0.1805	0.2	0.0005	1	64
0.1808	0.4	0.0001	1	256
0.1809	0.1	0.005	1	128
0.1818	0.3	0.0005	1	64
0.1858	0.0	0.005	1	256
0.1872	0.1	0.0005	1	32
0.1875	0.2	0.0001	1	128
0.1904	0.1	0.001	1	64
0.1912	0.2	0.005	1	256

(a) 104OYE

μ(loss)	d	lr	l	units
0.8205	0	0.00001	8	32,16,32,128, 16,64,32,256
0.8375	0	0.005	1	256
0.8607	0.2	0.005	1	128
0.8642	0	0.00001	8	32,32,256,256, 32,64,128,64
0.8657	0.1	0.001	1	256
0.8709	0	0.00005	8	128,256,256,16, 32,64,16,64
0.8833	0	0.00001	8	16,32,32,64, 64,32,64,64
0.8939	0	0.0001	4	32,16,32,256
0.8991	0.2	0.005	1	256
0.9058	0.2	0.0005	1	128

(b) 103HOE

μ(loss)	d	lr	l	units
0.9438	0.4	0.0001	8	16,16,128,256, 256,128,32,32
0.9473	0.4	0.005	2	64,256
0.9697	0.3	0.005	2	32,16
0.9750	0.4	0.005	2	256,128
1.0348	0.4	0.005	2	32,64
1.0495	0.4	0.005	2	32,128
1.0653	0.3	0.005	2	32,128
1.0737	0.4	0.001	2	32,32
1.0818	0.4	0.005	8	32,64,64,32, 128,128,32,128
1.0927	0.3	0.005	4	256,16,32,64

(c) 201D

μ(loss)	d	lr	l	units
1.3433	0.4	0.00005	8	64,256,32,32, 256,16,32,64
1.4334	0.4	0.0001	8	16,128,256,256, 64,256,64,16
1.5015	0.4	0.0001	4	16,64,16,16
1.5318	0.4	0.001	8	16,16,128,16, 16,256,128,128
1.5346	0.4	0.0001	8	32,256,128,64, 64,256,32,256
1.5786	0.4	0.00005	4	32,32,64,64
1.5955	0.3	0.00005	8	128,128,256,16, 128,16,16,256
1.6408	0.3	0.00005	8	64,128,256,128, 32,16,32,64
1.6452	0.3	0.00005	8	64,128,256,32, 32,128,16,256
1.6523	0.3	0.00001	8	128,32,16,128, 256,32,256,32

(d) 102BFS

Table C.1: MLP tuner results (dropout abbreviated as d, learning rate as lr and number of layers as l). Best 10 runs are shown. All runs are averages of 5 different models.

C.2 Quantile regression: RNN in multivariate setting

μ(loss)	T	d	lr	l	units
0.1996	G	0.1	0.0005	1	256
0.2017	G	0.1	0.005	1	64
0.2081	G	0.1	0.005	2	128,256
0.2112	G	0	0.001	1	256
0.2130	G	0.1	0.005	1	32
0.2178	G	0.1	0.0001	1	64
0.2180	G	0.1	0.00005	2	256, 128
0.2182	G	0.1	0.005	1	32
0.2232	G	0.1	0.0005	1	32
0.2240	G	0.1	0.00005	4	128, 128, 128, 256

(a) 104OYE

μ(loss)	T	d	lr	l	units
0.8506	L	0	0.005	1	256
0.8820	L	0	0.005	4	128, 32, 64, 256
0.9204	L	0	0.001	4	64, 64, 128, 128
0.9241	L	0	0.0005	4	128, 16, 16, 64
0.9279	L	0	0.005	4	16, 16, 128, 32
0.9304	L	0	0.0005	4	256, 16, 256, 64
0.9514	L	0	0.0005	2	32, 64
0.9547	G	0.3	0.005	4	64, 256, 256, 16
0.9726	G	0	0.0005	4	256, 32, 16, 128
0.9733	G	0.2	0.001	4	256, 64, 128, 128

(b) 103HOE

μ(loss)	T	d	lr	l	units
1.2741	L	0.4	0.001	4	64, 64, 64, 128
1.2755	L	0.3	0.001	4	32, 128, 32, 256
1.3064	L	0.4	0.001	4	128, 16, 64, 256
1.3563	L	0.3	0.001	4	64, 128, 16, 256
1.3641	L	0.1	0.005	4	64, 256, 128, 32
1.4079	L	0.3	0.001	4	64, 128, 16, 256
1.4101	L	0.2	0.005	4	16, 16, 32, 64
1.4211	L	0.3	0.0001	4	128, 256, 32, 128
1.4215	L	0.2	0.0005	4	32, 256, 32, 128
1.4250	L	0.1	0.005	4	128, 16, 128, 128

(c) 201D

μ(loss)	T	d	lr	l	units
1.0163	L	0.2	0.0001	4	32, 32, 128, 256
1.0259	L	0.3	0.0001	4	64, 32, 256, 128
1.0435	L	0.2	0.005	4	16, 256, 128, 128
1.0458	L	0.3	0.0005	4	128, 64, 64, 128
1.0572	G	0	0.005	4	64, 16, 256, 16
1.0635	L	0.3	0.0001	4	32, 32, 64, 256
1.0684	L	0.2	0.0005	4	32, 16, 64, 32
1.1015	L	0.2	0.001	4	64, 128, 128, 128
1.1102	G	0.3	0.0005	4	64, 64, 32, 128
1.1084	G	0.3	0.005	4	256, 32, 16, 16

(d) 102BFS

Table C.2: RNN tuner results (dropout abbreviated as d, learning rate as lr and number of layers as l, RNN type as T, with values L for LSTM and G for GRU). Best 10 runs are shown. All runs are averages of 5 different models.