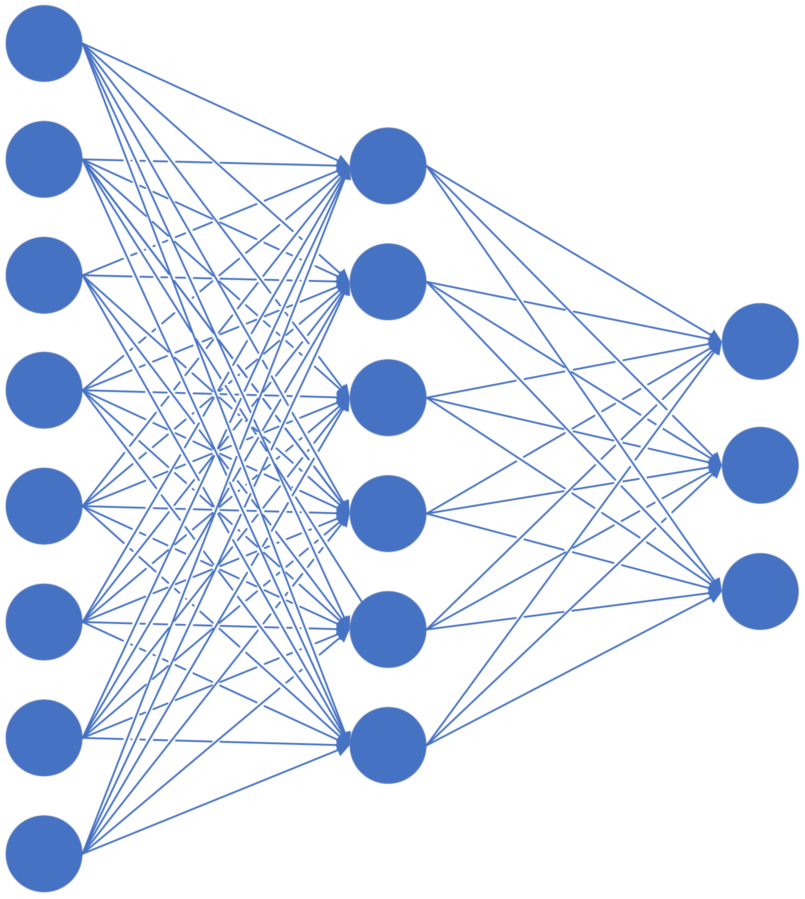
FMH606 Master's Thesis 2022

Industrial IT and automation

Specifying a machine learning operational framework, refinement and scaling of machine learning models for progressive cavity pumps at Den Magiske Fabrikken



Martin Holm

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis, 2022

Title: Specifying a machine learning operational framework, refinement and scaling of machine learning models for progressive cavity pumps at Den Magiske Fabrikken

Number of pages: 129

Keywords: Machine learning operations, machine learning, progressive cavity pump, predictive maintenance

Student: Martin Holm

Supervisor: Carlos Pfeiffer, Håkon Viumdal

External partner: **Lindum AS**

Summary:

One process part of "Den Magiske Fabrikken" uses progressive cavity pumps (PCP), these often have to be replaced which is costly. To reduce the amount of replacing and increase repairing with spare parts, a system to warn the operators about faults on these pumps is wanted.

The project uses machine learning in the form of long short-term memory (LSTM), gated recurrent unit (GRU) and support vector machine (SVM) to make models which will be able to predict when the pumps need maintenance. The models attempt to output whether the pump is *normal*, or when there is one week before general failure or when one day before general failure. It does so based on general measurements such as control signal, current, torque and outlet pressure measured every 30 seconds.

A model which from the test set never gives false warnings, 50% of the samples warns that there is one week until failure and 31.2% of the samples warn that there is one day until failure has been developed. The thesis also discusses how to operationalize such a model at a process plant.

Preface

This report is written as a Master Thesis in the 6th semester of the Industry Master Industrial IT and Automation master program. The report is a culmination of three years of work in the Industry Master program at Lindum AS and will give an overview for Lindum of what has been done. In addition, the work during the Master thesis should be useful to peers within the machine learning community for predictive maintenance in the process industry.

For the vision of this project and great support during it, I would like to thank Bertil Johansen, Jørgen Eikjeland for technical discussion and support, and Frode Steen for support during the thesis. I would also like to thank USN for having the Industry Master program, especially Carlos Pfeiffer and Håkon Viumdal for great support before, and during the thesis period.

This project uses Python, important libraries are scikit-learn, TensorFlow, Keras, pandas, Plotly and psycopg2 and their underlying dependencies. In addition, some plots are made in Microsoft Excel. Microsoft Visio is used for drawings and the report is written using Microsoft Word.

The Python code can be found on Github at:

https://github.com/martinsgugge/mast_predictive_maintenance_PCP

Basic knowledge of process industry, programming and machine learning will help the interested reader in understanding the contents of this report.

Tønsberg, 18.05.2022

Martin Holm

Contents

Nomenclature

Symbol	Unit	Description
PU		Pump
E		Container or vessel
FT	m^3/h	Flow transmitter
PT	Bar	Pressure transmitter
VY		Heat exchanger
V	Boolean	Valve
PUXX	Boolean	Control signal for pump XX
MO_PV	%	Control signal for pump
SF_PV	%	Speed of pump
PW_PV	A	Current measurement from pump
TQ_PV	%	Torque measurement from pump
Δt	Time	An arbitrary time from one point to another
Raw		Set of measurements: Control signal [Boolean], Control signal [%], Current [A], Torque [%] and Outlet Pressure [Bar]
Avg		Set of averaged measurements: Control signal [%], Current [A], Torque [%] and Outlet Pressure [Bar]
Std		Set of standard deviation of measurements: Control signal [%], Current [A], Torque [%] and Outlet Pressure [Bar]
Calc		Calculated efficiency measurements: Control Signal/Current, Control Signal/Outlet Pressure, Control Signal/Torque and Current/Outlet Pressure

Abbreviations

P&ID - Piping and instrument diagram

PLC - Programmable Logic Controller

VM - Virtual Machine

OPC UA - Open Platform Communications Unified Architecture

SCADA - Supervisory Control And Data Acquisition

ML - Machine Learning

MLOps - Machine Learning Operations

DevOps - Development Operations

AI - Artificial Intelligence

UP - Unified Process

OO - Object Oriented

LSTM - Long Short-Term Memory

GRU - Gated Recurrent Unit

SVM - Support Vector Machine

SVC - Support Vector Classifier

CI/CD - Continuous Integration and Continuous Delivery

ONNX - Open Neural Network Exchange

PI controller - Proportional Integral controller (Part of PID controller)

1 Introduction

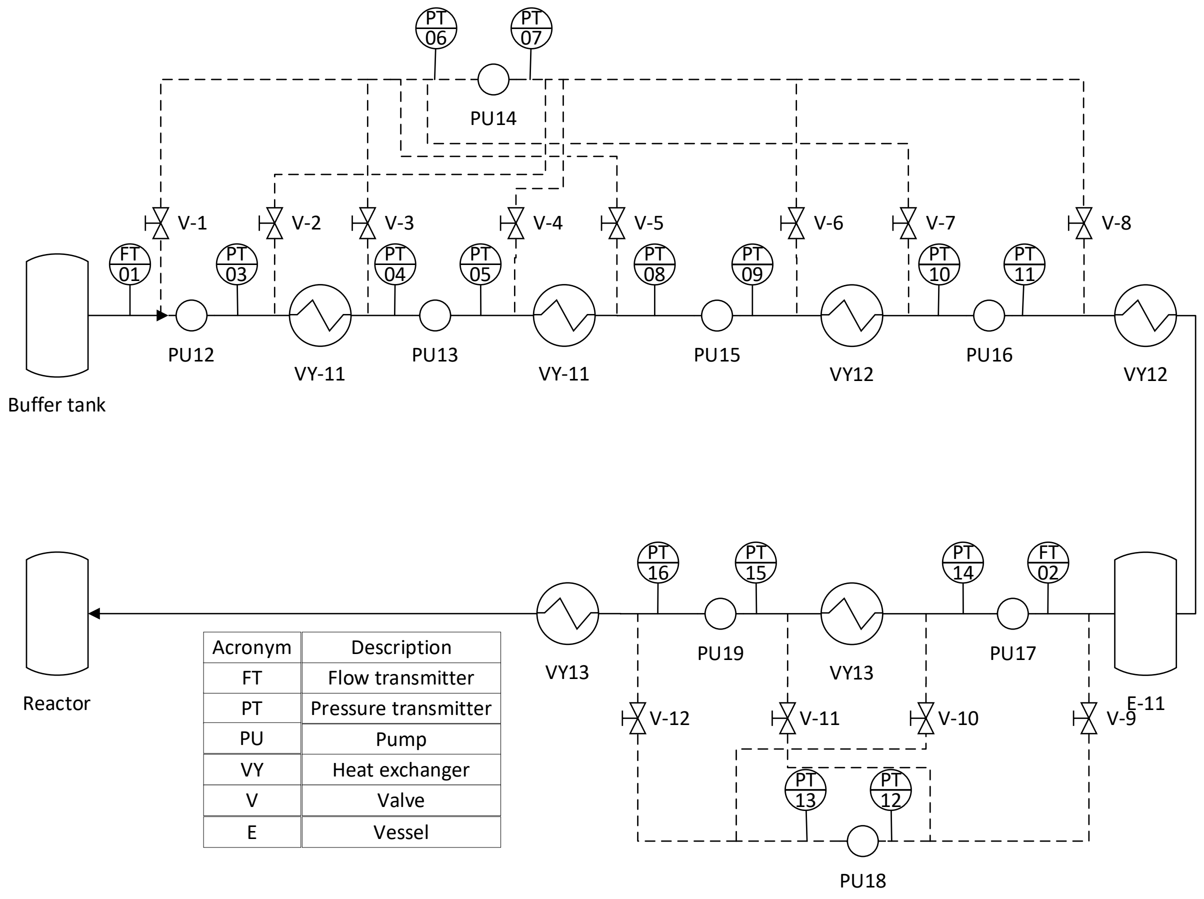
Lindum owns various factories and sites, amongst them two biogas plants. The report focuses on Den Magiske Fabrikken in Tønsberg, Norway. Specifically on a system of pumps where pumps are replaced every 3-60 weeks. They are replaced either when the operators feel it is time after inspecting the pump by listening to, touching the pumps or when they are unable to deliver wanted pressure. This approach creates a lot of unplanned work and often replacement of whole pumps instead of specific parts. Lindum wants to investigate whether the lifetime of the pumps can be extended by use of spare parts by having a model tell the operators when something is wrong. This may also alleviate stress from ad-hoc maintenance by allowing the operators to plan when to overhaul the pumps. The next subchapters will explain the systems which are currently in place, the objectives in this thesis, which methods will be used to achieve the objectives, and finally the report structure for the remainder of the report.

1.1 Existing systems

1.1.1 Process control system

The process system in question can be seen in Figure 1.1. The flow comes in from the buffer tank, is pumped through a series of heat exchangers where it gains temperature, before being pumped into a processing tank, E-11, where the substrate stays for one hour. When time is due, the substrate is pumped out of E-11, cooled down and pumped into the biogas reactor.

For each of the pumps several measurements are sampled from the control system and frequency converters, these can be seen in Figure 1.2. In the figure, PUXX can be any of the pumps seen in Figure 1.1 and the other rows in Figure 1.2 below are additional measurements which describes the pump in more detail.



Acronym	Description
FT	Flow transmitter
PT	Pressure transmitter
PU	Pump
VY	Heat exchanger
V	Valve
E	Vessel

Figure 1.1 P&ID where the pumps are situated

Tag	Description
PUXX	On/off signal
MO_PV	Control signal [%]
SF_PV	Speed [%]
PW_PV	Current [A]
TQ_PV	Torque [%]

Figure 1.2 Description of sub tags in pumps

1.1.2 Supervisory control and data acquisition system

The existing Lindum owned system can be categorized as a data acquisition and storage system in the orange blocks of Figure 1.3. The field level represents all the sensors and actuators at the plant, these are connected to the PLCs controlling the plant in the control level. The orange blocks contain the system which will be the main focus of this thesis.

The first orange block contains a Windows virtual machine (VM) which runs an OPC UA server called KepServerEx with a Datalogger addon [1]. This VM is considered the communication level. The Datalogger feeds data to the storage level. The storage level runs on virtual machine with the Linux distribution called Ubuntu, this is the second orange block. On the Ubuntu machine a PostgreSQL server runs with the addon TimescaleDB [2]. TimescaleDB gives more functionality for time-series operations and also allows for faster ingest and egress of data compared to a standard PostgreSQL database.

From the database there are two visualization platforms in the two lower orange boxes: one on the production network called Grafana [3] using the same Ubuntu VM as the storage level. Another is found on the Lindum network called QlikSense. Grafana is a query-based visualization tool, this means it requests data from the database when the data is needed. This contrasts the way QlikSense works, QlikSense loads all the data the user wants on a set interval, this makes the data available quicker when the user want to see it, however it requires more resources to ensure quick loading times.

In teal is the original SCADA system, which the operators use to control the plant.

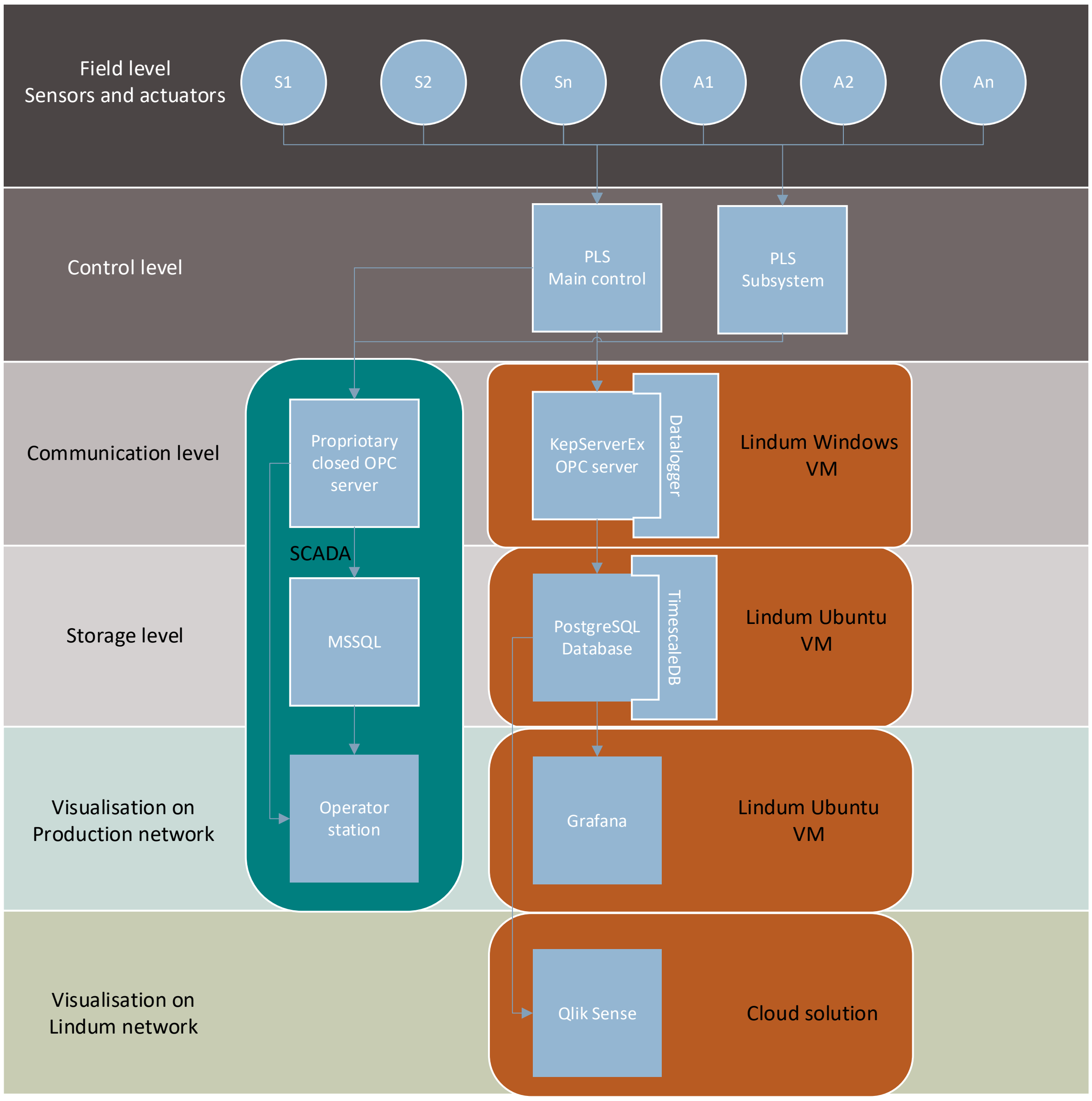


Figure 1.3 Supervisory control and data acquisition (SCADA) system including field level and visualization levels. Orange blocks are Lindum owned.

1.2 Previous work on predictive maintenance of progressive cavity pumps

This subchapter will describe some of the earlier work done in the field of predictive maintenance of the pumps. Two reports have been made: one on machine learning methods and fitting the data to several types of machine learning (ML) models, and another considering the design of a general software architecture for predictive maintenance. These will be summarized in section 1.2.1 and 1.2.2 to give an idea of where this project should continue.

1.2.1 Machine Learning for Predictive Maintenance of pumps at Den Magiske Fabrikken

The project “Machine Learning for Predictive Maintenance of pumps at Den Magiske Fabrikken” focused on trying out various ML models with three data sets where the main difference lies in measurement interval [4]. The variables can be seen in Table 1.1 along with a description and whether the data was part of each of the three data sets or not. The variables were sampled redundantly where it is part of more than one data set.

Table 1.1 Tags with descriptions and information on which tags are included in which data sets

Tag	Description	30 sec interval	1 sec interval	50ms interval
PU19	On/off signal	On change	On change	On change
PU19_PW_PV	Current [A]	Yes	Yes	Yes
PU19_TQ_PV	Torque [%]	Yes	Yes	Yes
PU19_MO	Control signal [%]	Yes	Yes	Yes
PU19_SF_PV	Speed [%]	Yes	Yes	Yes
PT15	Pressure before PU19 [Bar]	Yes	Yes	Yes
PT16	Pressure after PU19 [Bar]	Yes	Yes	Yes
FT02	Flow from processing tank E-11 [m^3/h]	Yes	Yes	Yes
PU19_V_L	Velocity on bearing house [mm/s]	No	Yes	No
PU19_V_I	Velocity inlet [mm/s]	No	Yes	No
PU19_V_O	Velocity outlet [mm/s]	No	Yes	No
PU19_P_L	PeakVue bearing house	No	Yes	No
PU19_P_I	PeakVue inlet	No	Yes	No
PU19_P_O	PeakVue outlet	No	Yes	No

With the assumption that the process variables would change as the pump became more and more degraded, five states or classes were made and can be seen in Table 1.2. During the project it was found that the state *stopped* (0) was unnecessary, and that states *less than 24h before failure* (3) and *Less than 1h before failure* (4) were very hard to predict. The latter two may have been because of the bias towards *normal running* (1) and the *less than one week before failure* (2) being too great due to the difference in amount of training data. Another possibility is that the states 2, 3, and 4 may have been too similar and thus hard to distinguish.

Table 1.2 Output classes for the pump model

State	Description
0	Stopped
1	Normal running
2	Less than 1 week before failure
3	Less than 24h before failure
4	Less than 1h before failure

The data was plotted in various ways to look at how the variables were distributed, how they correlated with each other and to see whether the correlations were linear. It was found that some variables had high correlation especially the current and torque when averaged. The relationship between inlet pressure and flow and the relationship between inlet pressure and control signal were sometimes shown to be nonlinear. See chapter 2 of the report [4] for more details.

Several ML methods were tested with varying results as seen in Table 1.3. The leftmost column indicates the used data set, and the top row indicates the method. The principal component analysis (PCA) was able to describe much of the variations, however it was difficult to clearly distinguish *failing* from *normal running*. The higher the resolution, the easier it was to distinguish, although the explained variance did not increase. The Short-Time Fourier Transform (STFT) gave some additional information to the rightmost ML methods (Naïve Bayes (NB), Support Vector Machine (SVM), Long short-term memory (LSTM)), however the non-averaged data sets should have been used as these may have given more information. The Naïve Bayes gave rather poor predictions and may be due to the variables being highly correlated and the Support Vector Machine didn't do much better. This may be due to the changes happening gradually and thus it may be hard to find good separations. LSTM gave better results, the 30 second data set gave the best results. This is likely due to there being 13 pump failures for this data set and only three for the 1 s data set and two for the 50ms data set.

A big change occurred when moving from scaling data from all the pumps simultaneously to scaling each pump with the first hour it was running. Accuracies then increased; this was only done for the LSTM method.

Table 1.3 Results from different combinations of data sets and methods

Data set/Method	PCA [explained variance]	Short-Time Fourier Transform on torque [Yes/No] (Used in NB, SVM and LSTM)	Naïve Bayes [Accuracy]	Support Vector Machine (Radial Basis kernel) [Accuracy]	Long short-term memory [Accuracy]
30 sec	79.62%	Yes	54.9%	55%	55%
30 sec (state 0 removed)	84.95%	No	N/A	N/A	N/A
30 sec new scalar per pump	No	Yes	No	No	78.5%
1 second averaged to 20s	75.36%	Yes	27.5%	45.9%	No
1 second averaged to 20s scaled per pump	No	Yes	No	No	71.8%
50 ms averaged to 1 s	77.33%	Yes	16.6%	17%	No
50 ms averaged to 1 s scaled per pump	No	Yes	No	No	59.5%

1.2.2 An architectural design for implementing predictive maintenance in an industrial plant

This article focused on an architectural design for a general predictive maintenance system for the process industry. A centrifugal pump with vibration measurements is used in conjunction with normal pump measurements (such as power, flow and pressure) in the ML method as an example. The machine learning model here is a self-organizing map (SOM). SOM is enticing as it is unsupervised, and at the point when the process measurements aren’t linked to specific faults, it can be used to show the difference between normal and bad states.

The point of the article is however not to choose what sensors and methods that should be used. It is rather about the process of setting up a predictive maintenance scheme, what to consider when training a model, using supervised or unsupervised learning, how labelling should be done for industrial equipment and what software modules should exist and what the software modules should do. Figure 1.4 shows a flow diagram from the proposed architecture taken from the article. Key takeaways are: the structure of the architecture, that the functions should be modular to allow for other pre-processing and machine learning methods to be used, and the contribution over time block. The contribution over time will slow down the process, but also make it less erratic, meaning a false positive will not have a big impact unless there are several false positives in a time interval. This does however reduce the chance of picking up rapid changes. This is only a summary, a detailed review of the architecture can be found in [5].

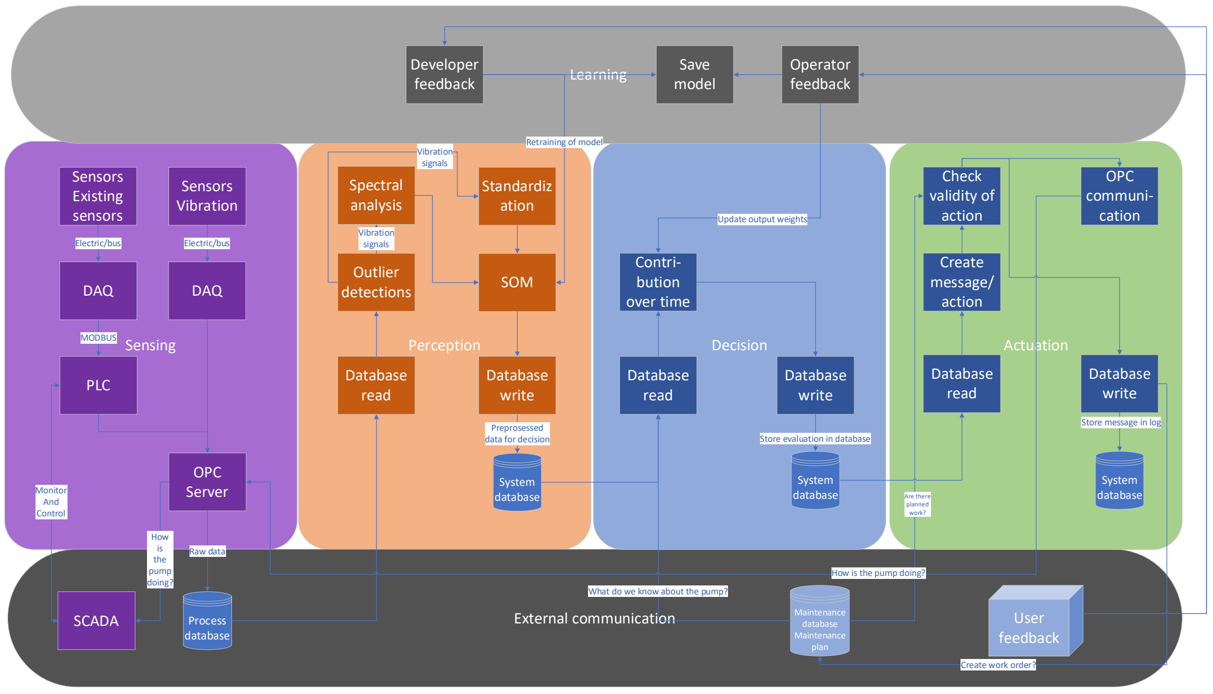


Figure 1.4 Flow diagram of data processing for a ML program in an industrial plant. Taken from the article [5].

1.3 Project objectives

This thesis will have two areas of interest, understanding what is needed for operationalizing a machine learning model, and modify the model by generalizing for all the 8 pumps seen in Figure 1.1.

To understand how to operationalize the model, MLOps will be investigated, and the inception phase of the Unified Process will be used to describe the needs. The inception phase may be used to decide whether to buy a product or to develop the program inhouse.

On the point of improving the model there are several points:

- There are more or less useless outputs from the model made, the “stopped” state is already known from the SCADA system. Also, the output “1h from failure” was never predicted in the test which was outside the training and validation set in any of the models. Thus, these classes may be skipped to simplify the model.
- Trying out new methods.

- Changing parameters of the models such as number of layers, number of neurons and network structure.
- Using more data from the other pump locations.
- Change input features for the models.

The model should be applicable to other pumps than pump 19 (HYG_PU19), there are additionally 7, very similar pumps, the model should be able to monitor them as well.

1.4 Machine Learning Operations

As a machine learning model is developed, it will eventually need to be set into production to give value. Once in production it needs to be monitored and maintained. This workflow is called machine learning operations or MLOps. MLOps is defined as *"the extension of the DevOps methodology to include Machine Learning and Data Science assets as first class citizens within the DevOps ecology"* [6] [7]. DevOps doesn't have a consensus on a definition, however a suggestion has been *"a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality"* [8]. For the purpose of this report, MLOps is a framework on how to effectively and orderly form a machine learning project from start to finish modularly. This ranges from finding features in the data, creating data pipelines, versioning data, features and models, model training, evaluation, deployment, and monitoring. Continuous integration will not be a high priority but may be a result of the methodology.

AI readiness "denotes a company's maturity in incorporating AI into the business" [9]. There are three levels of AI readiness for an organization, tactical, strategic and transformational. Tactical is the lowest level, where all tasks for ML is manual. Strategic is one level up where some tasks are automated, typically pipelines for pre-processing and model training. Transformational is the highest level of AI readiness where most tasks are automated [9].

1.5 Software engineering

The ML framework needs some way of being expressed, here the Unified Process (UP) will be used to define the specifications of the needed software. Unified Process is a framework to design software for object-oriented (OO) programs and constitutes the whole development process of the given software [10]. The UP consists of 4 main phases: Inception, Elaboration, Construction and Transition phase. This report will only investigate the inception phase and elaboration phase. The interested reader for the whole process may find more information in [10] or [11].

The Inception phase decides whether the project should be executed or not by making a high-level plan on how to execute the project. It also decides on how the project team should be set up and what the business cases are for the project. Lastly it may decide whether to purchase or develop the software in house[1].

Elaboration phase starts to identify use cases, define functional and non-functional requirements and the software architecture. If it is decided to develop the software in-house, an implementation of the architecture may also be developed and tested.

[1] The decision to buy or develop can also be made in the elaboration phase

One way to describe the functional and non-functional requirements is to divide them into the attributes Functional, Usability, Reliability, Performance, Supportability and Other (FURPS+) [10].

- Functional - Capabilities of the system.
- Usability - How does a human interact with the software and how much training is needed.
- Reliability – Expected uptime, mean time between failure
- Performance – Response time, data throughput, resource usage
- Supportability – Configuration, operating system, new algorithms
- + – Resource limitations, interface constraints (to systems)

1.6 Machine learning

This subchapter will go through the main methods of machine learning used in this thesis.

1.6.1 Long short-term memory (LSTM)

Long short-term memory, often known as LSTM, is a type of recurrent neural network which has the capacity to "remember" earlier states. This enables the network to see sequential connections and see when the sequence may differ compared to only seeing what happens now as an independent try. Comparing to a standard recurrent neural network, LSTM does not struggle as much with vanishing or exploding gradients during training [12]. LSTM is good at solving issues related to sequential data and is typically used for speech recognition, handwriting recognition and time-series forecasting. Figure 1.5 shows an LSTM block, these are typically set up in series to create "memory" in between samples in a sequence. The LSTM memory may persist between sequences, yet most problems may be solved by using stateless LSTM, that is no memory between sequences only within a given sequence [13].

The LSTM block is a combination of four neural networks. These are there to control three *gates* which regulate how much information should be forgotten (forget gate), added (input gate) and outputted (output gate) from the *cell state*. The cell state is the memory previously mentioned which is the core of an LSTM network.

The forget gate uses a neural network with sigmoid activation function to decide how much information should be forgotten from the cell state for each timestep. The sigmoid network takes in the previous output timestep, h_{t-1}, and the current input timestep, x_t, of data as one vector. Based on the training, the network will know which parts to remove from the previous cell state C_{t-1}. The network outputs a value between zero and one. Zero meaning forget everything of the given memory, and one, keep all the information about the given memory. After calculation of the sigmoid network, the results are multiplied with the previous timestep's cell state, C_{t-1}.

The input gate uses a combination of a neural network with sigmoid activation and a neural network with tanh activation funciton. The sigmoid network work decides in what capacity the previous output timestep, h_{t-1}, and the current input timestep, x_t, should be added the new cell state, C_t. Zero meaning do not add and one meaning add all that the tanh layer gives. The tanh layer also takes h_{t-1} and x_t, to create a potential new cell state which will be multiplied with the output of the sigmoid layer. This potentially new cell state are vectors similar to the cell state with values between -1 and 1 and decides in what direction and magnitude the cell state should be updated. The outputs of these two networks (sigmoid and tanh) are multiplied to create a vector which will be added to the previous cell state C_{t-1} to create the current cell state C_t.

The output gate consists of a sigmoid network, a tanh function and a multiplication of these. The sigmoid network takes h_{t-1} and x_t to decide what part of the cell state should be outputted. The tanh function takes the cell state C_t to squish the values to be between -1 and 1 and outputs the product of the sigmoid network and the squished cell state, h_t.

As mentioned, Figure 1.5 is an LSTM block, this will be executed for each timestep in the sequence iteratively and the cell state, C_t, and output, h_t, will be transferred to the next iteration. Traditionally for each of the iterations there will be an actual output such that there will be a sequence as output as long as the sequence input [14]. Optionally there can be one output from the whole input sequence. The number of neurons will be the same for all the neural networks inside the LSTM block.

For a more thorough description, some variations on LSTM and the math behind see [15] and [12].

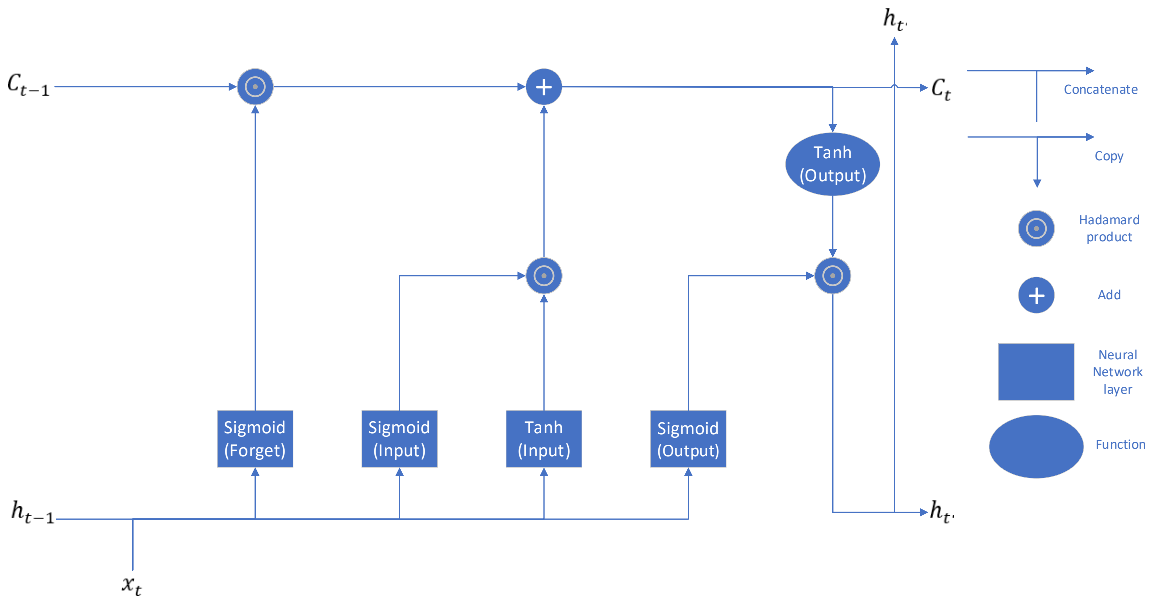


Figure 1.5 LSTM block[2]

1.6.2 Gated Recurrent Unit (GRU)

Gated Recurrent Unit (GRU) is a variation of Long Short-Term Memory (LSTM) and thus also a recurrent neural network. GRU has been found to cost less computation and give similar results to LSTM [16]. The structure of a GRU block can be seen in Figure 1.6 where there are fewer neural networks (rectangular boxes) comparing to the LSTM, this reduces the number of parameters to tune. This makes for two gates, reset and update gates. In addition, there is a tanh neural network for the candidate hidden state.

The reset gate decides how much of the hidden state should be added to the candidate hidden state h'_t based on the previous hidden state h_{t-1} and the current input x_t. This is done by passing h_{t-1} and x_t to a neural network with a sigmoid activation function to get the reset

[2] Hadamard product; Element-wise product of two matrices [44]

vector r_t which works similar to the forget gate of LSTM, a zero value removes the memory and a one keeps the memory, it may also be partially forgotten. The Hadamard product of h_{t-1} and r_t decides how much of the hidden state should be kept for the candidate hidden state h'_t.

The candidate hidden state is a neural network with tanh activation function and takes in the modified previous hidden state from the reset gate and the current input. The output is a value between -1 and 1 and is used to update the next hidden state (and output).

The update gate also uses a neural network with sigmoid activation function to decide what parts of the previous hidden state h_{t-1} and the candidate hidden state h'_t to keep. The output of the network gives a vector z_t. The Hadamard product of z_t and h_{t-1} decides what part of the hidden state is kept, while the Hadamard product of 1-z_t and h'_t decides what part of the candidate hidden state is kept. Finally, the two Hadamard products are added together to give out the final output, h_t, of an iteration [17] [18].

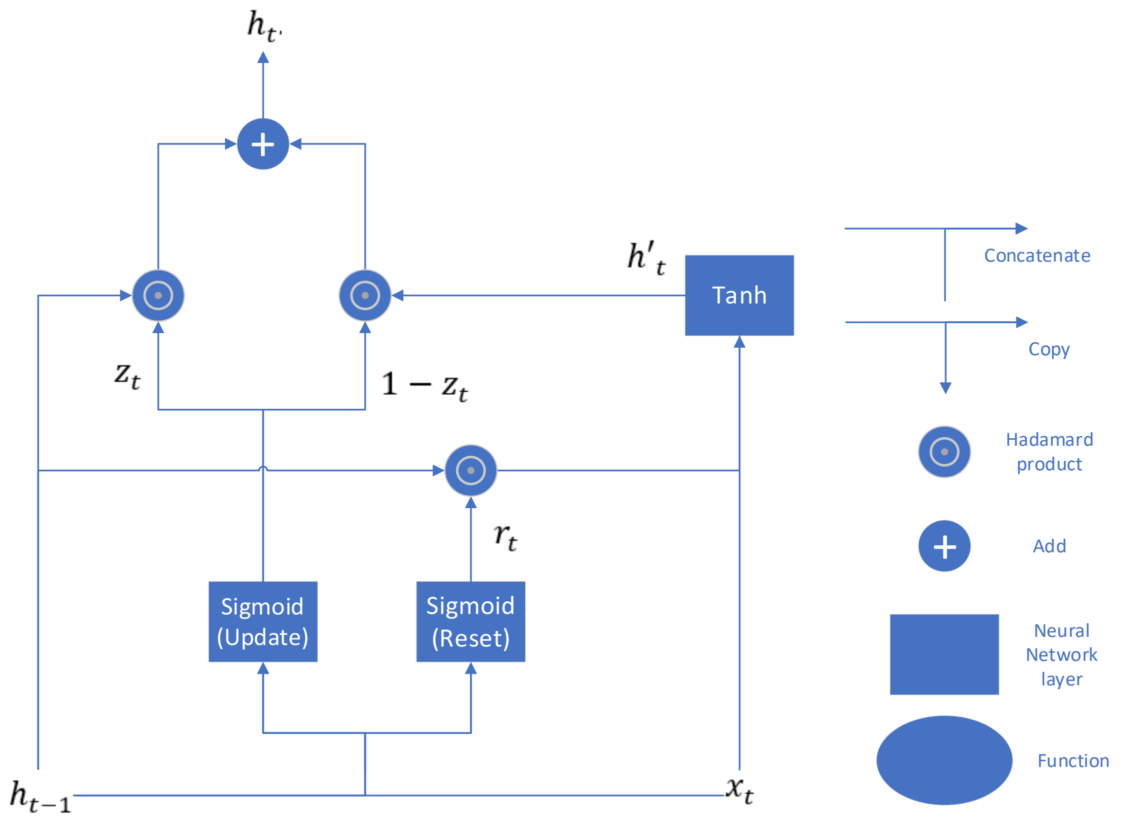


Figure 1.6 GRU block

1.6.3 Neural network complexity

There are no hard rules on how to decide how many layers and how many neurons should be in each layer for a neural network [19]. A single layer may suffice to model close to anything, yet it may not be as efficient as a network with more layers. Complex problems may profit from having deeper networks. The number of neurons in each layer also adds to what complexity the model can handle. Gradually increasing can be a good way to find the right amount. One source recommends using equation (1) where N_h is the number of neurons, N_i is number of inputs and N_o is number of outputs [20].

$$N_h = \frac{2}{3}(N_i + N_o) \tag{1}$$

1.6.4 Support Vector Machine (SVM)

Support vector machine is a supervised machine learning method used for classification. It attempts to put data points into classes by drawing a line (support vector) which maximizes the Euclidian distance between the training datapoints which are labelled as different classes [21]. Originally the support vectors were linear, later the method has been modified to allow for nonlinear vectors by using kernel functions. The kernel functions are a proxy for projecting the data into higher dimensions where the data may be split linearly. When returned to "normal" dimensions, the linear separation from the higher dimension folds and becomes nonlinear. Originally it was a binary classifier but has been modified to allow for multiple classes.

There are several variations to SVM, most notably hard and soft margin. Hard margin either creates an optimal hyperplane if it exists or fails to create a hyperplane to use as a margin. The hard margin does not allow for samples to be misclassified during the training and works best where there is a clear distinction between the classes. Hard margin does not allow for samples to be misclassified in the sense that there are mathematical constraints which must hold, where the margin (hyperplane in higher dimension) must divide the samples according to the classes. If not, the method fails.

Soft margin on the other hand allows for some error in its optimization problem where the number of errors can be minimized. This allows for classification where the samples are somewhat overlapping between classes.

The original SVM were made to linearly divide two sets of classes. This obviously allows only for linear separation and has since been remedied by the use of kernel functions. The idea is to push the data into higher dimensions where the data may be linearly separable, find a hyperplane which separate the classes. This is computationally heavy and is in practices not done. In stead a kernel trick is done where the nonlinear boundaries are calculated directly based on the kernel function. Popular kernel functions are linear, radial basis function, sigmoid and polynomials of arbitrary degrees [22].

The SVM is in its nature a binary classifier, it can however be used as a multiclass classifier. This is done by changing the multiclass problem into several binary problems. The most common ways to implement this is with the one-versus-all and one-versus-one methods. One-versus-all checks whether this classifier (one of the several binary problems) scores the highest. The one-versus-one uses voting where every classifier will output one of two classes, the class with the highest number of votes, wins.

Figure 1.7 shows a three-class problem solved by a linear kernel. It uses soft margins as the data overlaps to a large degree as seen by the crossover between the different colors i.e., white is in the red, light blue and dark blue areas.

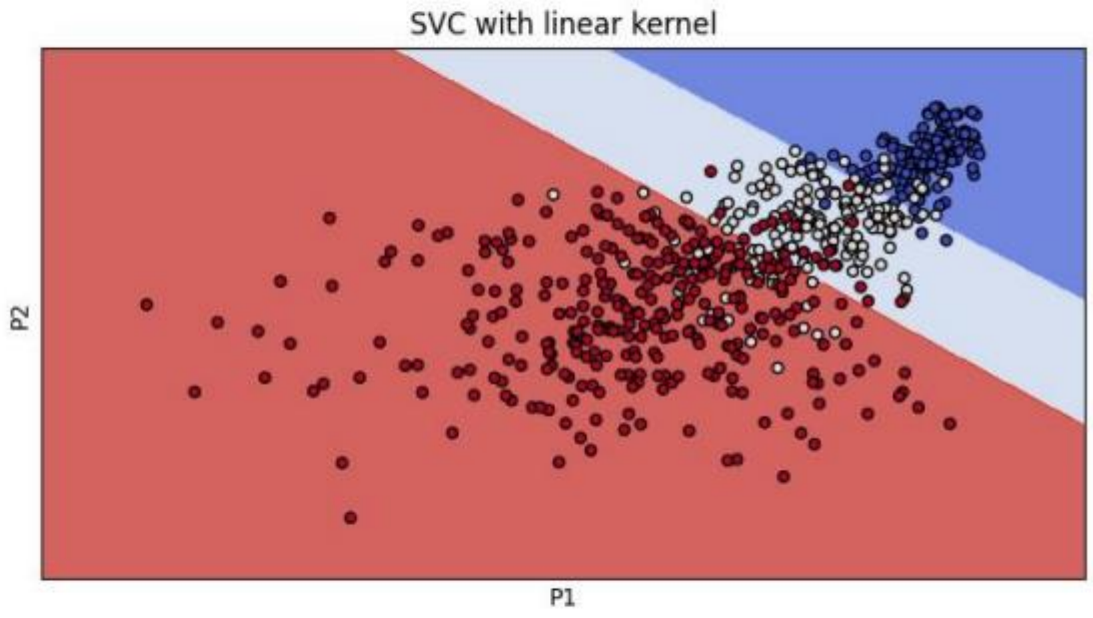


Figure 1.7 Three class SVM problem

Comparing to the recurrent neural networks LSTM and GRU, SVM is a lightweight method with the potential to classify just as well. This was seen in a previous project where SVM and LSTM was tested, initially they both seemed to give rather good results on the validation sets [4]. Only when tested on an entirely new data set was it revealed that none of them performed very well. On a second attempt with the LSTM, the data was scaled as explained in section 1.6.5, the LSTM model performed significantly better. The scaling method should be reproduced here for the SVM.

1.6.5 Scaling

In the previous project explained in section 1.2.1, the scaling method was found to be of high impact on the LSTM model. The scaler used was a standardization, which "*removes the mean and scales to unit variance*" [22]. Figure 1.8 shows the difference in the scaling method, the here called old method uses one standardization for many physical pumps. The new method creates one standard scaler for each new physical pump. This specifically had a high impact on testing outside the training area. Both SVM and LSTM gave good results before changing to the new scaling method for the test set within the training area but gave bad results when testing on an entirely new physical pump failure. This new scaling method was tested on LSTM with good results. It is thus not unreasonable to consider that SVM also can give good results when using the new scaling method. A detailed review of the scaling method can be seen in Appendix B (draft for publication).

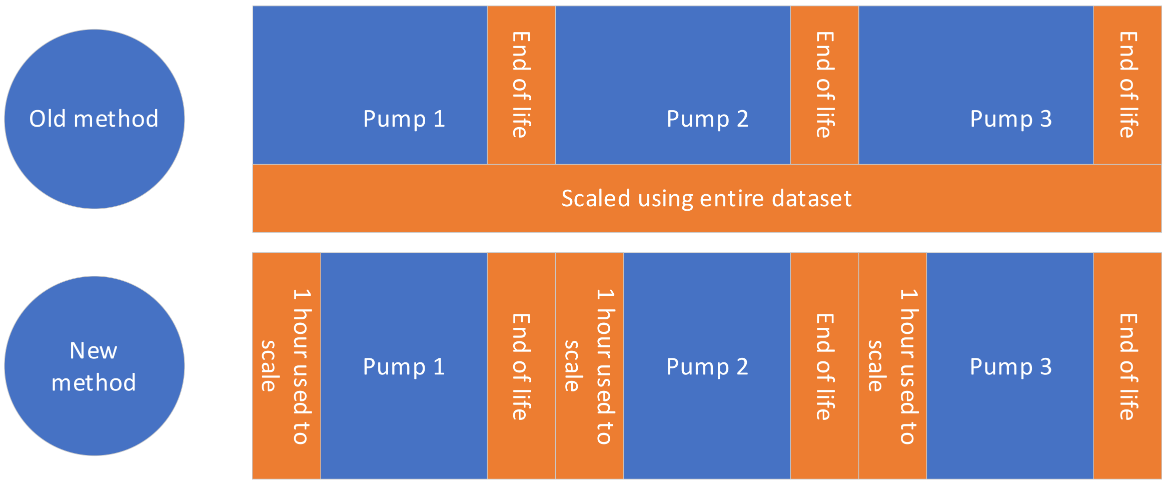


Figure 1.8 Difference in how the data was standard scaled. Image from Appendix B.

1.7 Report structure

Further on this thesis will investigate the use of machine learning operations in chapter two. This serves to create a software specification and get input on how to run a machine learning project. The third chapter gives the results for the software specification. In chapter four, the thesis starts to focus on the machine learning part where the data pipelines for training are explained and experiments are defined. The fifth chapter shows the results from the models developed during the experiments. Chapter six discusses alternatives on the models, what could have been done differently to indicate what should be done in future work. Chapter seven concludes the work done in the thesis.

2 Software specification

As machine learning models are being developed a system for them to run in will be necessary. This chapter will investigate what the standards of machine learning operations are, as well as the needs in an industrial sense for the specific use case of predictive maintenance. In essence, the model will need to receive data, process it, run it through a model and finally sending the inferred data to someone or something which can use the data as seen in Figure 2.1. First, best practices on how to develop and deploy machine learning models will be investigated. This serves three purposes; Using best practices in the machine learning development process later in the project, to understand how models are deployed and the relationship between the two.

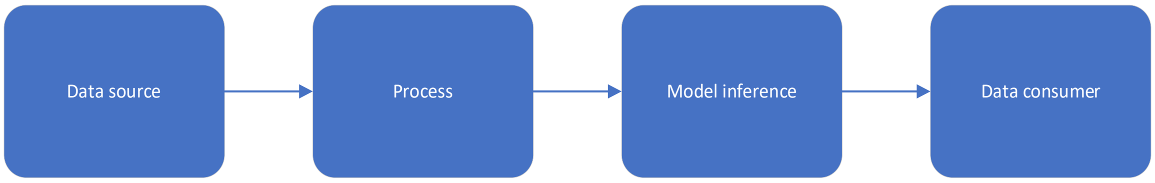


Figure 2.1 High level data flow for a deployed machine learning model

2.1 Machine learning operations

Machine learning operations (MLOps) can be seen as a continuum as mentioned in section 1.4. It may be strategic to not aim for the highest level of MLOps in the first round of machine learning development in a company. An assessment based on the size and experience of the team which works with machine learning within the company may be useful as well. To do this, the MLOps stack canvas can be used [9]. The MLOps stack canvas will be filled out with information from the next subchapters and can be seen in Figure 2.2.

2.1.1 Data sources and data versioning

The process data sources available are a PostgreSQL database and an OPC server on the production network of the plant. The operation logs which are written by the operators daily for information about when a pump has been replaced are available in an online file server. The process data is readily available on the production network; however, it may be simpler to work with the data online, thus a backup should be taken to an online computer. The operation logs are written in free-text and on a form of morning shift and evening shift, on a daily basis with notation Monday-Sunday with a corresponding week. Thus, data about when the pumps are replaced must be extracted, together with the date from week-day combination. Finally, the dates must be matched with a time in the process data to find when the pumps are replaced. When a pump is replaced, the backup pump is started. From the combination of one pump stopping and the backup starting, the timestamp of the replacement can be found.

Data versioning is likely not very necessary as time-series data is not expected to change once logged. It may be useful to know which time frames are used to train the models for historical reasons, more on this in section 2.1.6.

2.1.2 Data analysis & experiment management

The initial data analysis has been done as described in section 1.2.1. Python was used for the initial analysis, and python will be used further on. Initially, the project seems feasible with the best model giving a 78.5% accuracy. Other evaluation methods may be investigated for the models such as F1-score, precision and recall. To process more data GPU processing power should be available as training deep neural networks (DNNs) such as long short-term memory (LSTM) and gated recurrent unit (GRU) may take a very long time on a CPU. On low interval data such as 50ms, only 5minutes of every hour was used in training the LSTM model, the data was also averaged to 1s which reduced the processing needs 20-fold. To train a model with all the available data and not using average requires a lot more processing power (or time).

To manage the experiments, several parameters for each type of model should be stored. Amongst them are an identifier, features, pre-processing steps, training time, what data has been trained on, test accuracy, network configuration for the LSTM and GRU models, kernel function for SVM and hyperparameters depending on the kernel used.

2.1.3 Feature store and workflows

The feature store allows for extracted features to be shared from data engineers to ML model engineers. This allows for a feature to only be found once, then stored such that it can be reused for other training sessions easily. One of the points of doing this is to avoid the ML model engineers having to use any low-level API to get the data. As the project for now is close to a solo project and all tasks are done by one person, there is little need to make hard rules and layers to standardize features for a “next step”. In the live serving of the model, the data will still be received from a SQL server and must be pre-processed with a similar pipeline and use the predict method instead of fitting method. Transformations can be logged in the database, such as to avoid calculating the transformations every time the model is trained again.

2.1.4 Foundations of DevOps

As the IT department is rather small, there is next to no DevOps at Lindum. Lindum uses GitHub, this is used primarily to hold code, versioning and collaborating on the code. The code may run on offline servers which makes continuous integration and continuous delivery (CI/CD) hard. Systems may be monitored by feeding data to a database through the firewall, although this is not implemented. Metrics on deployment frequencies, lead time for change, mean time to restore, and failure rate is not stored.

2.1.5 Continuous integration, training and deployment

As there is next to no DevOps in Lindum, automating integration, training and deployment is not feasible. There may not be much need as Lindum does not use many models yet, and the models will likely not change fast. As the number of models increase, the point on continuous integration, training and deployment should be reevaluated together with resources appointed to these tasks.

Training may occur inside the production network, on an online computer or in a cloud environment. The workflow for of the pipeline in development is dependent on manually taking data out from the production network, finding useful features, pre-processing it, training and producing a model before evaluating. A production pipeline should be similar; however, it should automatically get the data, use the features which has been found useful, process and run through model and serve results. For development the Python programming language has been used with primarily Pandas, SciPy and TensorFlow. Further on Kedro may be used for pipelines and AirFlow may be used for scheduling. Distributed model training is not yet considered necessary.

Non-functional model requirements:

Fairness, the training data is rather skewed at least in the way the networks has previously been set up as seen in Table 1.2. When basing the output categories on time from failure where the “*normal* operations” time frame is much larger, the training data unless modified will also have a much larger proportion of *normal* operations. This may not be a problem assuming that the outputs have group fairness in the form of equalized odds [23]. This indicates that the categories (the group) have or have close to equal true positive rate (equalized odds).

2.1.6 Model registry and model versioning

The model registry and versioning component is as the name implies used to register models, this can range from having control on what data the model has been trained on, which is newer, which is better in various metrics. An important note is also to be able to go back to a previous model. The model registry is a useful tool to keep track of the models, especially when the number of models start to grow. The models can be stored on a file server, or on the server it is running from and details about each model can be described in a database or CSV file until more resources are available.

2.1.7 Model deployment

The model will be delivered as an Open Neural Network Exchange (ONNX) file which “is an open format built to represent machine learning models” [24]. The expected time for changing the model should be on the scale of months as the pumps are typically replaced with an interval between 3-60 weeks today. Thus, retraining of a single model should not be expected to be more often than one month. One month may even prove a high interval, as there are likely not much changes happening. The target environment will depend on the solution to run the model, some solutions require to be on-premise others are more flexible or require to be on cloud.

2.1.8 Prediction serving

Prediction serving is the act of giving new unknown data to the model and receiving a result. There are generally two ways of prediction serving: online and batch mode [9]. There are five patterns for serving, the most applicable ones are Model-as-Service, Model-as-Dependency and Precompute. Model-as-Service wraps the model as an independent service and can be accessed through a REST API [25]. Model-as-Dependency includes the model inside the software and thus makes the software dependent on the model. The Precompute pattern computes data batchwise, stores the predictions in a database where users can query for results.

As the predictions are wanted cyclically, Precompute may be the best option. Thus, i.e., one hour of data can processed in a batch and the results are stored in a database where other programs may query for the data. As briefly discussed in section 1.2.2, a contribution over time may be useful to reduce the number of false positives, this is to reduce the potential erratic behavior of independent trials.

How many trials should be included in such an average for it to be trustworthy, yet still be reactive to what is happening will have to be trial and error when a sufficiently good model has been achieved. A trustworthy model indicates that the users trust the model when the model gives a warning. If not trustworthy, it may quickly fall out of fashion and not be used, the trustworthiness is thus essential.

As the model is attempting to predict errors one week ahead of time, but at the same time give the operators time to plan work, the model should serve predictions between every hour and every day. As it is not very costly to predict, once an hour may be sufficient.

2.1.9 Model, data and system monitoring

Monitoring of the model may be hard as the goal of the model is to avoid the pumps being damaged beyond repair. If the pumps aren't run until failure anymore, how is it known whether there was one week left or one day left? The assessment of whether the model predicts the right label or not will to some degree be a subjective matter from the operators as they are the ones inspecting, repairing or replacing the pumps. The predictions will of course be logged, a procedure or system must be defined to allow the operators to give their feedback.

There are some reasons why the data would change, the pump type may change, there will always be variations in the dry matter, some new equipment may be installed causing a change. While the pump type may change, this can also be a known factor, and may be considered a different model. By knowing which type of pump is installed, operators may change the model used for a tag[3]. The dry matter will invariably change and must be built into the model, this will influence the pump's behavior. Sensors can be installed to get this information, however then a new model may need to be trained.

The deployment system should also be monitored, and mainly consist of connections monitoring of database and/or OPC servers.

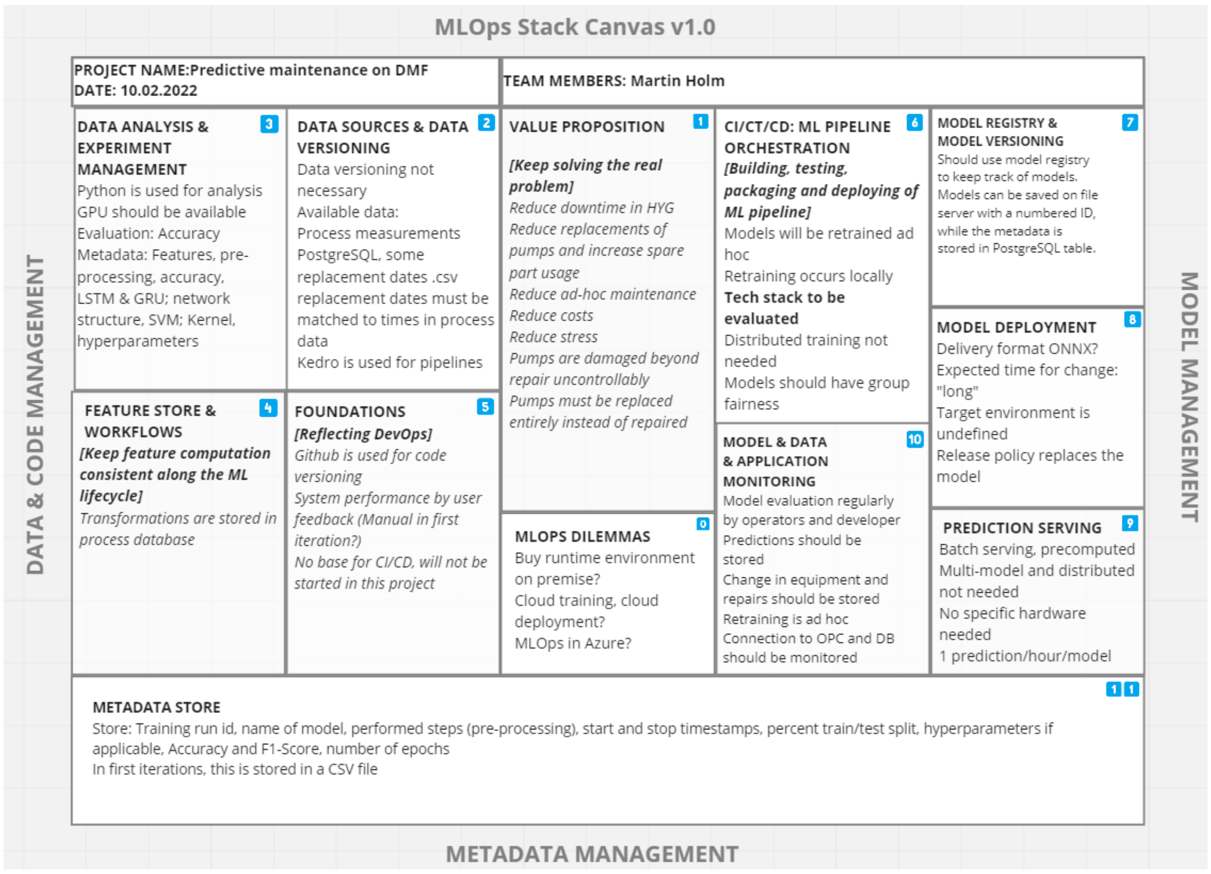
2.1.10 Metadata Store

The metadata store is used to store information about the models developed. This data can include what features and time frames are used, how it is pre-processed, hyperparameters of models, various evaluation results and more [9].

2.2 Software scope

To run the model automatically a system for getting the data, transforming it to the right format, processing it, inferring the information from the model and sending the information to an operator or to another software is needed. The software does not include training the model as this may be considered a separate step in MLOps. The previous section on MLOps goes through some steps related to data needed, the processing of the data and deployment of the model as well as metadata that should be known and monitored. Figure 2.2 shows the culmination of the sections on MLOps and from this a scope can be defined.

[3] Tag – A position within a process control system, physical equipment may be changed, but the positions name remains the same



MLOps Stack Canvas v1.0

PROJECT NAME:Predictive maintenance on DMF
DATE: 10.02.2022

TEAM MEMBERS: Martin Holm

DATA & CODE MANAGEMENT

DATA ANALYSIS & EXPERIMENT MANAGEMENT 3
Python is used for analysis
GPU should be available
Evaluation: Accuracy
Metadata: Features, pre-processing, accuracy,
LSTM & GRU; network structure, SVM; Kernel, hyperparameters

DATA SOURCES & DATA VERSIONING 2
Data versioning not necessary
Available data:
Process measurements PostgreSQL, some replacement dates .csv
replacement dates must be matched to times in process data
Kedro is used for pipelines

VALUE PROPOSITION 1
[Keep solving the real problem]
Reduce downtime in HYG
Reduce replacements of pumps and increase spare part usage
Reduce ad-hoc maintenance
Reduce costs
Reduce stress
Pumps are damaged beyond repair uncontrollably
Pumps must be replaced entirely instead of repaired

CI/CT/CD: ML PIPELINE ORCHESTRATION 6
[Building, testing, packaging and deploying of ML pipeline]
Models will be retrained ad hoc
Retraining occurs locally
Tech stack to be evaluated
Distributed training not needed
Models should have group fairness

MODEL REGISTRY & MODEL VERSIONING 7
Should use model registry to keep track of models.
Models can be saved on file server with a numbered ID, while the metadata is stored in PostgreSQL table.

MODEL DEPLOYMENT 8
Delivery format ONNX?
Expected time for change: "long"
Target environment is undefined
Release policy replaces the model

FEATURE STORE & WORKFLOWS 4
[Keep feature computation consistent along the ML lifecycle]
Transformations are stored in process database

FOUNDATIONS 5
[Reflecting DevOps]
Github is used for code versioning
System performance by user feedback (Manual in first iteration?)
No base for CI/CD, will not be started in this project

MLOPS DILEMMAS 0
Buy runtime environment on premise?
Cloud training, cloud deployment?
MLOps in Azure?

MODEL & DATA & APPLICATION MONITORING 10
Model evaluation regularly by operators and developer
Predictions should be stored
Change in equipment and repairs should be stored
Retraining is ad hoc
Connection to OPC and DB should be monitored

PREDICTION SERVING 9
Batch serving, precomputed
Multi-model and distributed not needed
No specific hardware needed
1 prediction/hour/model

MODEL MANAGEMENT

METADATA STORE 11
Store: Training run id, name of model, performed steps (pre-processing), start and stop timestamps, percent train/test split, hyperparameters if applicable, Accuracy and F1-Score, number of epochs
In first iterations, this is stored in a CSV file

METADATA MANAGEMENT

Figure 2.2 Machine learning operations stack canvas [9]

Starting with the data source for inference is the process database, the system will need an interface to the PostgreSQL database. Further on the data will generally need some pre-processing such as outlier detection, scaling and transformations, typically called a data pipeline. When the data has reached the end of the pipeline, it is used to infer information from the model. At this point, the information is ready to be consumed by applications or users. To get to a consumer, the inferred information needs to be transmitted in some way, this can be either to a database or to an OPC server. There are primarily two possible consumers, the operators using the SCADA system or the maintenance program for placing a work order when the pumps need to be fixed. A third alternative is to store it in a database where another application may use it. For the consumer to use the information, the data needs to be formatted in a way that suits both the transmission protocol and the consumer. As such the system not only needs interfaces to one or more of these consumers, but it also needs a translation layer from the outputs of the model to something that makes sense for the consumer. The general scope can be seen in Figure 2.3.

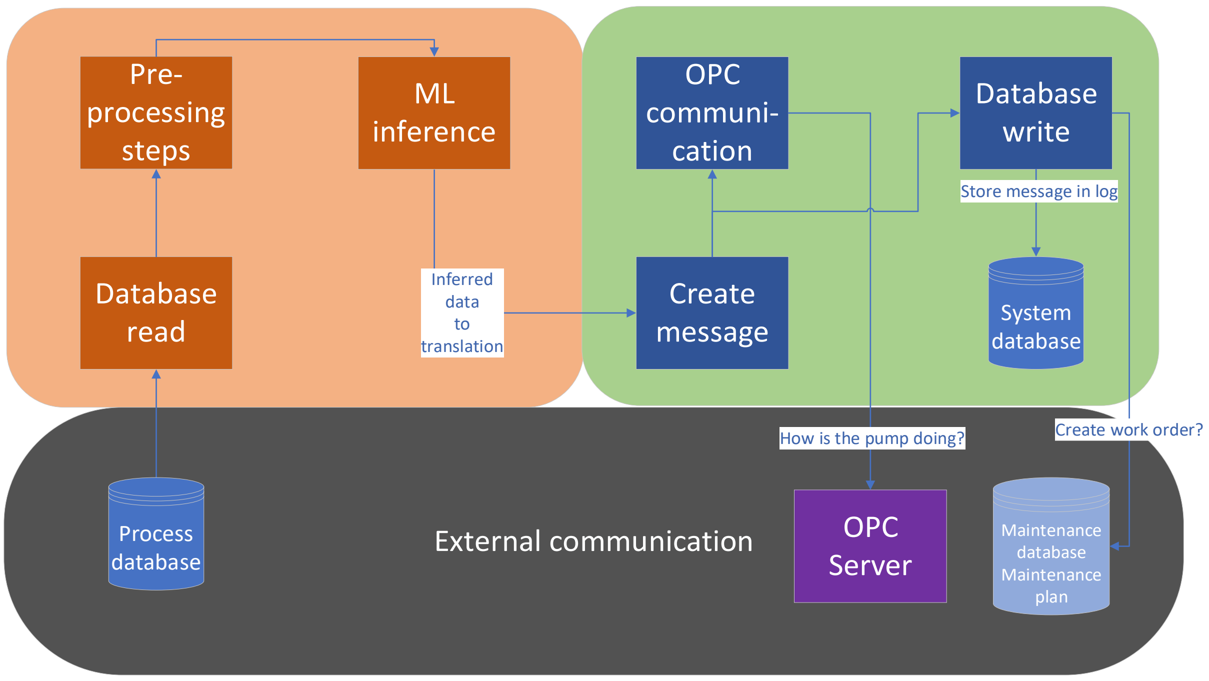


Figure 2.3 Scope for software to infer data from model

3 Software specification results

This chapter shows the resulting software specification.

3.1 Requirements

The requirements will attempt to concretely define what is needed from a software to run the model. Here there is a separation in the MLOps, training is not considered here as this may be done separately. The requirements may be achieved using a modular solution; thus, the system does not need to be one program. To frame the requirements FURPS+ is used. FURPS+ is Functional Usability Reliability Performance Supportability and other design challenges or limitations.

3.1.1 Functional

Run ML model files: As the system is going to use a machine learning model, it must be able to run it. At a minimum ONNX files must be supported, in addition pickle objects, TensorFlow objects .pb, PyTorch’s .pt are nice to have.

Communicate with OPC UA: the software needs to send data to the SCADA system this can be done using an existing OPC server. To do this, the software needs to have functionality for receiving OPC UA events with a subscription and be able to send data to the OPC UA server.

Communicate with PostgreSQL: The program needs to get process data from the PostgreSQL server, it also needs to send data to a PostgreSQL server to store results

Communicate with SQLExpress: The program needs to communicate with the maintenance management system to create work orders. Creating work orders likely will require to be able to create, read and update records.

Maintain configuration: As various models are developed, there is a need to switch out which models is used, data sources, how the data is processed (data pipelines). This may be done in either a database or in local files.

3.1.2 Usability

The user should view results from the system using the SCADA system or maintenance program. The user does not directly interact with the system as this is a cyclically called function which gives results.

3.1.3 Reliability

This is essentially not a critical system, thus an availability of 95% should suffice which equates to 18.26 days downtime per year [26]. The program should run batchwise when available and errors must be logged.

3.1.4 Performance

A single core CPU of 1GHz, 2GB RAM, 4GB of storage will suffice plenty for a single model of the size. This is based on running a test with three pump lifetimes. With more models comes higher requirements, although the specifications are already oversized.

3.1.5 Supportability

As different problems will require different solutions in the form of pre-processing and machine learning method there should be possibility to integrate Python scripts to do some of the work, thus allowing processing techniques and libraries to be used.

3.1.6 Other

Communication to the maintenance program and the SCADA system will require specific methods and formats. The maintenance program does not currently allow for external communication, however an API has been started on by a supplier. The SCADA system can be communicated with over OPC. The SCADA supplier will have to provide tags which the program can send information to.

3.2 Infrastructure

There are primarily two ways the model can be set up to run; on premise (local intranet), or in the cloud. There are advantages and disadvantages to both, this subchapter will investigate some of them.

3.2.1 On premise

Running the machine learning inference on the hardware which already exists may appear the most intuitive. The processing power, RAM and storage is already there. The machine holding these resources are however offline which creates a few complications. The first is updating, this becomes harder due to operating system variations from online computer to offline computers. Finding the correct dependencies to install a given program, finding the correct modules for i.e., Python or C# and their submodules. There is generally less automation when working offline which makes it more time consuming.

As have been discussed in section 2.1, much of DevOps and thus MLOps is based on reducing time from a change is initiated to it is deployed. There are frameworks to work from on this which can be installed locally. This does require some of setup and combining of various system LF AI & Data Foundation Interactive Landscape gives an overview of many solutions [27]. There also exists some solutions which are considered end-to-end MLOps which indicates that they cover more or less all the steps of MLOps [28].

Because the model is running inside the production network, it will not have to cross the firewall to receive and send data to/from the database and OPC server as seen in Figure 3.1. This removes one of the risk factors. It will still have to connect to the maintenance database, the maintenance database is on Lindum’s own network. For this communication, it may be enough that the ML model sends data out from the production network.

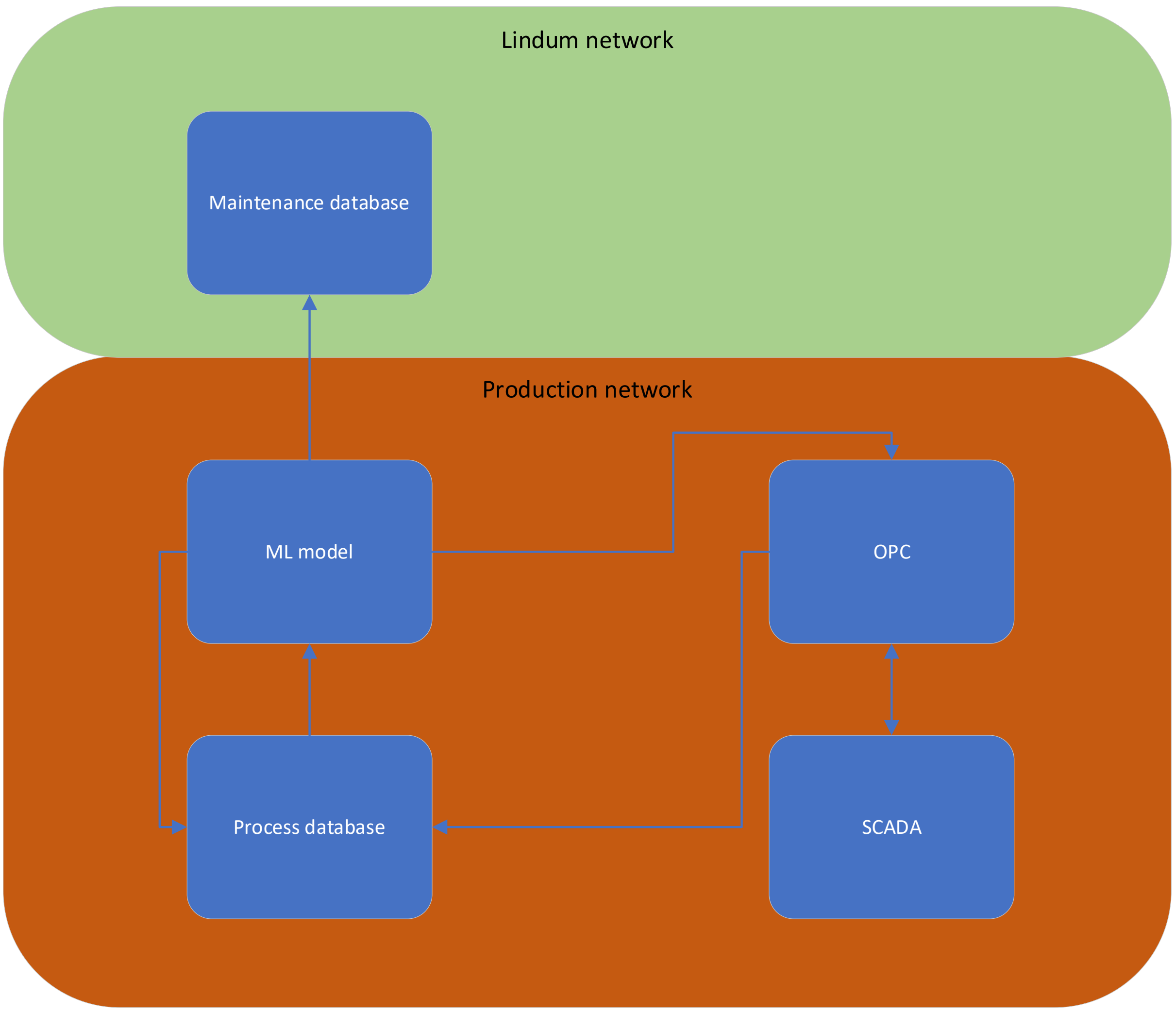


Figure 3.1 Running the model on premise

3.2.2 Cloud

Running the model in the cloud has the clear disadvantage that there are more firewalls to cross, thus exposing the production network at several locations as seen in Figure 3.2. It will also be an extra expense to process the inference of the model(s). It will however be easier to integrate the model to cloud MLOps solutions which would allow for easy updates of the models and software surrounding the model. Using the cloud will in general also allow the hardware to scale easily and for the infrastructure to run models for several factories. This will reduce the amount of maintenance.

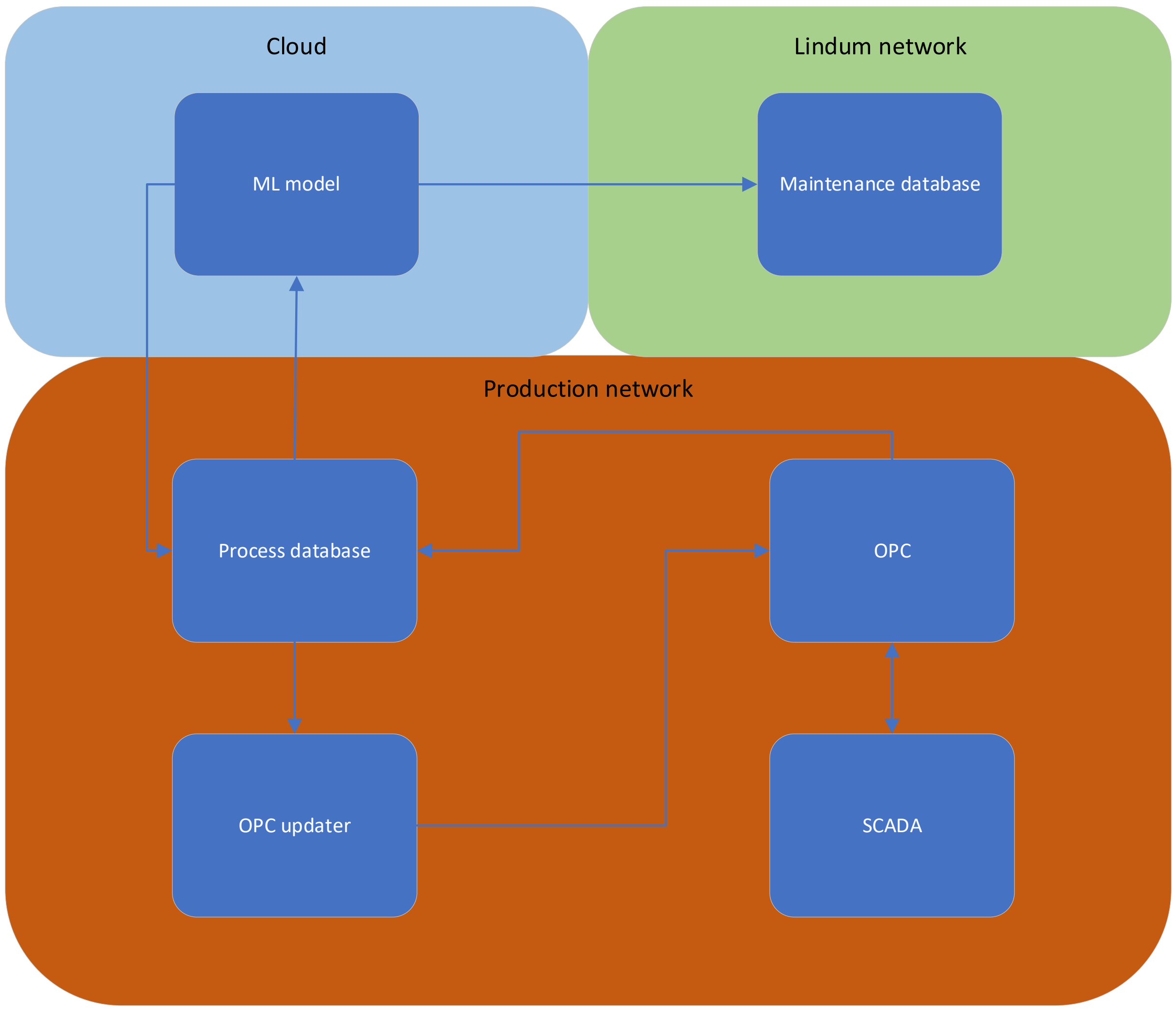


Figure 3.2 Using cloud to run the machine learning model

3.3 Risk

There are various types of risks, in broad they can be divided into safety and security risks. Safety risks are unwanted events which happen by mistake or negligence, while security risks are intentional by an attacker [29].

There are several cyber security risks to assess, no matter where the model is deployed data has to go through firewalls and open up to the production network (sending data in or out). The OPC server runs on the production network and is directly in communication with the PLC. The maintenance system is in Lindum's online network.

3.3.1 OPC

- The OPC server should maintain the current security level using certificates, difficulties have been observed in approving the certificate using Python API (in case of developing the system in-house).

- Should the model be run in the cloud, connecting directly to the OPC server may pose a cyber security threat as the OPC server can directly control the PLCs. Using certificates should be safe, yet additional measures may be taken, the database may function as an intermediate where an internal program on the production network queries the database and updates the OPC server. Using datatype checks, it should be very hard to maliciously control the OPC server.
- Writing to wrong tags may disturb production

3.3.2 Maintenance system

- The maintenance system runs on Lindum's network and needs connection to the model runtime.
- There are no available stored procedures to create a work order, this will have to be developed. Can be outsourced.
- Creating the wrong work order may issue work where it is not needed.
- Creating redundant work orders may cause confusion and frustration.

3.3.3 Model

As a machine learning model is not deterministic it cannot be guaranteed to give the same response for a given set of inputs. This does bring up the possibility for several iterations to give a wrong prediction and thus letting the equipment stay too long or for the equipment to be replaced too early. Remedies can go in the way of understanding how the model behaves once implemented and use that experience to read the model. Alternatively, the model can use an additional layer such as a moving average or checking how many of the previous samples are similar.

3.3.4 Operator feedback

If ultimately the operator needs to give feedback to the system, how should this be done? The operators should not be able to update the model directly, rather the model may be retrained on an ad hoc basis as the model likely does not need frequent retraining. The information about what is replaced should be stored and ideally an estimate of the state of what is replaced. I.e., is a seal broken, damaged or fine. Some procedure for assessing the equipment should be made.

4 Machine learning, data and use of methods

The second part of this thesis considers the training of several models, the data pipeline to get and transform the data, and evaluating the results of each experiment. The general data flow to train a machine learning model is similar to that of inferring information from one as was seen in Figure 2.1. Figure 4.1 shows the general dataflow for machine learning, first there is a need to find useful data, then there is often a need to pre-process the data before training a model and evaluating the model. This chapter will go through these steps to explain the paths taken to get the results that will be shown in section 5.

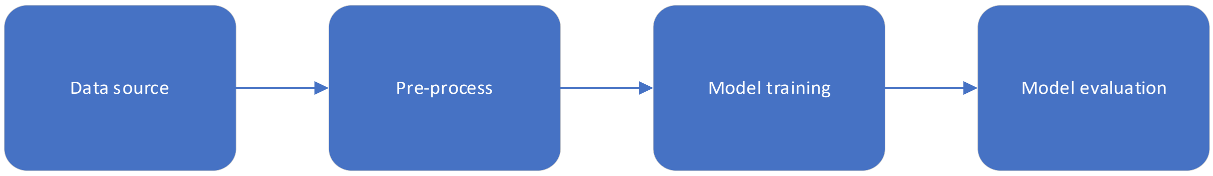


Figure 4.1 Machine learning flow diagram

4.1 Data source

The data source is a PostgreSQL database which is queried from a Python script. The data is stored in various tables and thus requires some logic to put it together into a coherent data set. Most process variables are stored in specified time intervals, Boolean process variables are stored when the value changes and the labels are stored when they change. Two data sets exist, one, which has been stored every 30 second and one which has been stored every 50ms. The 30 second data set has been stored with some outages from March 2020 and data to train with has been taken out up until February 2022. The remaining 30 second data from February to May will be used for testing. The 50ms dataset has been stored since July 2021, however due to limited time, it will not be used in this thesis even though it may hold valuable information. Figure 1.1 and Figure 1.2 best describes what process variables exist and a general model will be seen in section 4.3.1.

4.2 Data labelling

To use supervised learning, the outputs must be labelled. To label the data, the pump failure times are needed. When a pump is replaced or repaired, it should be written in an unstructured log. It is also possible to see when a pump is replaced in the process data by looking at which pumps are being used and which are not. As the pump failures and repairs are not always written down in the unstructured log, the process data needs to be investigated.

Figure 4.2 shows the Boolean control signal from one of the main pumps (pump 17) in blue and a backup pump (pump 18) in red. The signal is cyclically being turned on and off and can be used to indicate when a replacement or repair is happening. It is assumed that for each time a similar pattern occurs, a pump is being replaced. As there is no known way of telling the difference between an overhauled pump and a new pump, these are assumed to be the same.

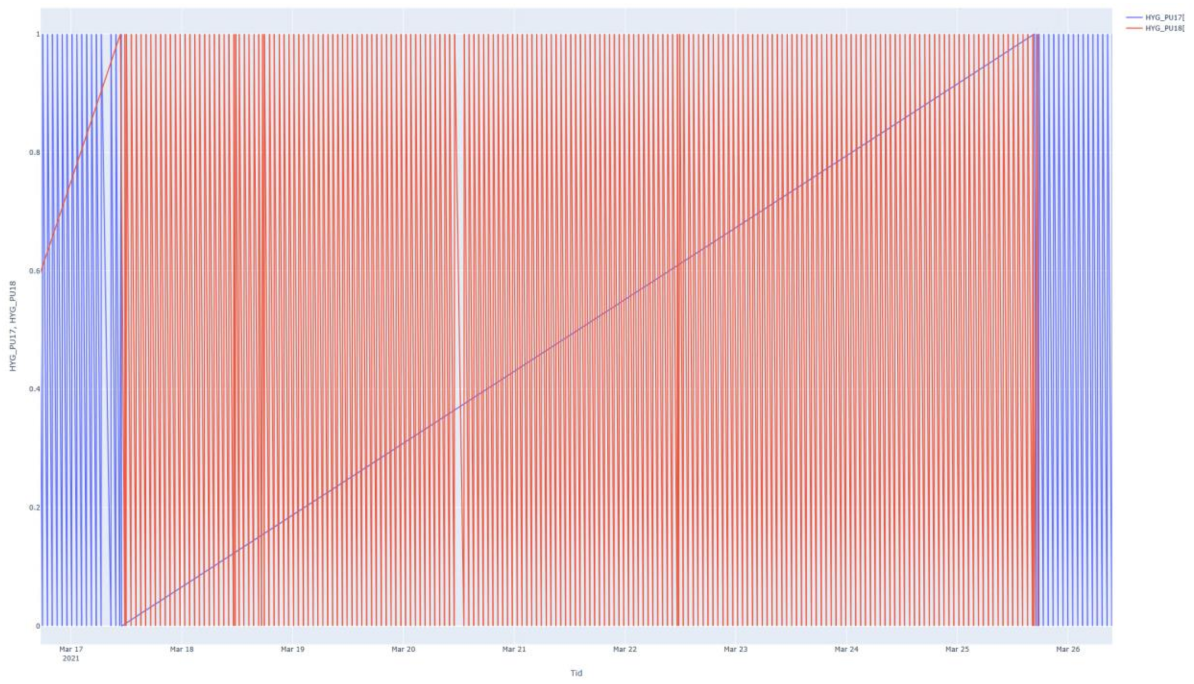


Figure 4.2 Time-series plot of Boolean control signal for pump 17 (main pump) in blue and pump 18 (backup pump) in red. When a pump is taken out of operation, the backup pump will activate which is seen here, the blue (pump 17) Boolean control signal stops cycling on and off 17th March and the red (pump 18) Boolean control signal starts to cycle on and off. This indicates that pump 17 has been deemed by the operators unable to do its job properly, and pump 18 is doing the work while pump 17 is being replaced or overhauled.

The pumps will be labelled in two ways, one to indicate what predictor state the pump is in, an extract can be seen in Table 4.1. The other to indicate when a "new"[4] pump is set into production again in

Table 4.2.

Comparing to the states from the previous project, two of the states have been removed. Table 1.2 shows the five states from the previous project. The first, "stopped" and the last, "1h to failure" were deemed unhelpful as "stopped" is already known and 1h to failure was never predicted. This leaves three states, *normal running*, *less than one day before failure* and *less than one week before failure* as seen in Table 4.1.

Table 4.1 Extract from pump state data, joined with labels for readability

Time	Tag	Status
14.10.2020 10:18	HYG_PU12_State	Normal running
07.02.2022 03:23	HYG_PU12_State	Less than 1 day until failure
01.02.2022 03:23	HYG_PU12_State	Less than 1 week before failure
08.02.2022 18:01	HYG_PU12_State	Normal running
14.12.2020 17:20	HYG_PU13_State	Less than 1 week before failure
08.12.2020 17:20	HYG_PU13_State	Less than 1 day until failure
16.12.2020 19:41	HYG_PU13_State	Normal running

[4] New, in this scenario is considered either an actual new pump, or a pump that has been overhauled and is put into production again

Table 4.2 Extract from pump number

Time	Tag	Pump number
07.07.2020 12:00	HYG_PU12_Pump_no	1
08.02.2022 18:01	HYG_PU12_Pump_no	2
16.12.2020 19:41	HYG_PU13_Pump_no	1
06.07.2021 10:21	HYG_PU13_Pump_no	2
13.08.2021 10:24	HYG_PU13_Pump_no	3

4.3 Feature selection

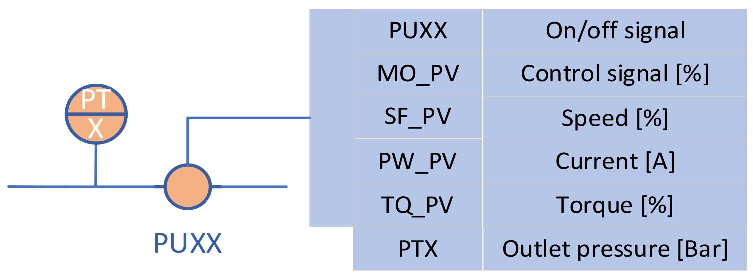
Feature selection is choosing what features, variables, sensors which should be used in the machine learning model. Choosing the features wisely can reduce training times, increase accuracy and reduce complexity of the model. The features can be many things, raw data, properties about the raw data, aggregates, or combinations such as fractions or multiplication of data. Raw data can give important details about local variations, aggregates can give longer term variations. The combination can reduce the number of connections (free parameters) the model needs to learn by itself [30] [31].

To get a feeling for the data, several plots will be made to see trends, correlations and other properties. This may help in understanding if any features can be dropped, what features can be extracted from the available data, what pre-processing steps that need to be done.

This may be important as there may be variations between the physical pumps and their location in the process. The idea here will be to investigate if it is feasible to use data from various pumps for one model to monitor all the pumps. It is especially interesting to investigate if they lie in the same general range after being scaled. In the project described in section 1.2.1, standardization was used based on the first hour of the pumps running time. The principle of using the start of a pump's lifetime is likely a good choice when considering that the data for the whole series will not be available when the model should infer on a real pump. Using this approach will require the pump to be running for the period used to create the scalers before inferring.

4.3.1 Base features

As the pumps are relatively similar, so is the sensors for the pumps. These eight pumps have 6 variables in common and can be seen in Figure 4.3. As was seen in Figure 1.1, the system of pumps before and after the processing vessel each has a flow transmitter (FT). If one model for each pump was created, this may have yielded good information. However, when creating one model for all the pumps, the flow measurement will have some time delay between the pumps as illustrated in Figure 4.4. The time delay is visualized as one Δt from the flow transmitter to PU17 and with an arbitrary number of times, x, Δt from the flow transmitter to PU19. The flow transmitter may be included in a potential future calibration phase using transfer learning. It can also be noted from Figure 4.4 that some pumps, such as PU19 has inlet and outlet pressure, while others only have outlet pressure such as PU17. As only some have the inlet pressure, the general model cannot have it.



PUXX	On/off signal
MO_PV	Control signal [%]
SF_PV	Speed [%]
PW_PV	Current [A]
TQ_PV	Torque [%]
PTX	Outlet pressure [Bar]

Figure 4.3 Common measurements for the progressive cavity pumps

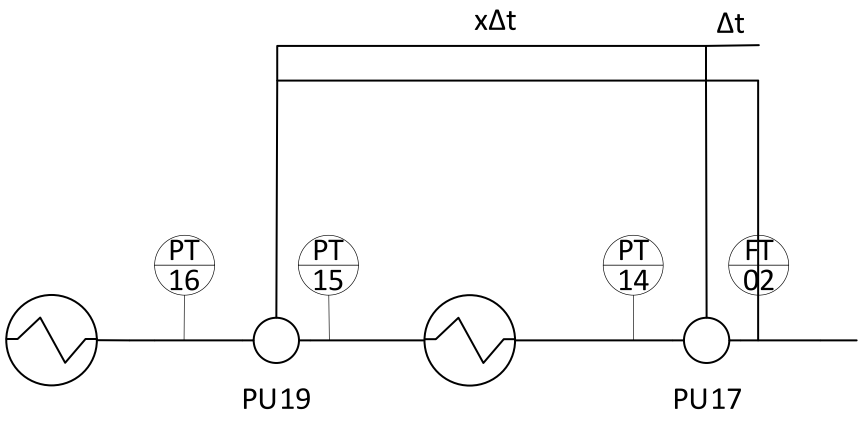


Figure 4.4 Time delay from flow transmitter to two different pumps

4.3.2 Specifics in the data set

There are two backup pumps which seldom are replaced, and it is hard to discern when they are replaced from the data available. From what is known from reports and process data, pump 14 has not been replaced during the data collection. Thus, when a scaler was created in the automatic way, it is based on the data seen in Figure 4.5. The control signal in red is stationary at five, which is not necessarily wrong, however the pressure in beige seems to change, even though the current (green) and torque (blue) is at zero. This would obviously confuse the model as the outlet pressure is varying while the pump is not running. As there has only been one of these pumps, it may be useful to leave this pump out of the training as there would require new logic to include it properly. This also goes for Pump 18 (HYG_PU18), it does have one replacement, yet in the interest of time it will be skipped.

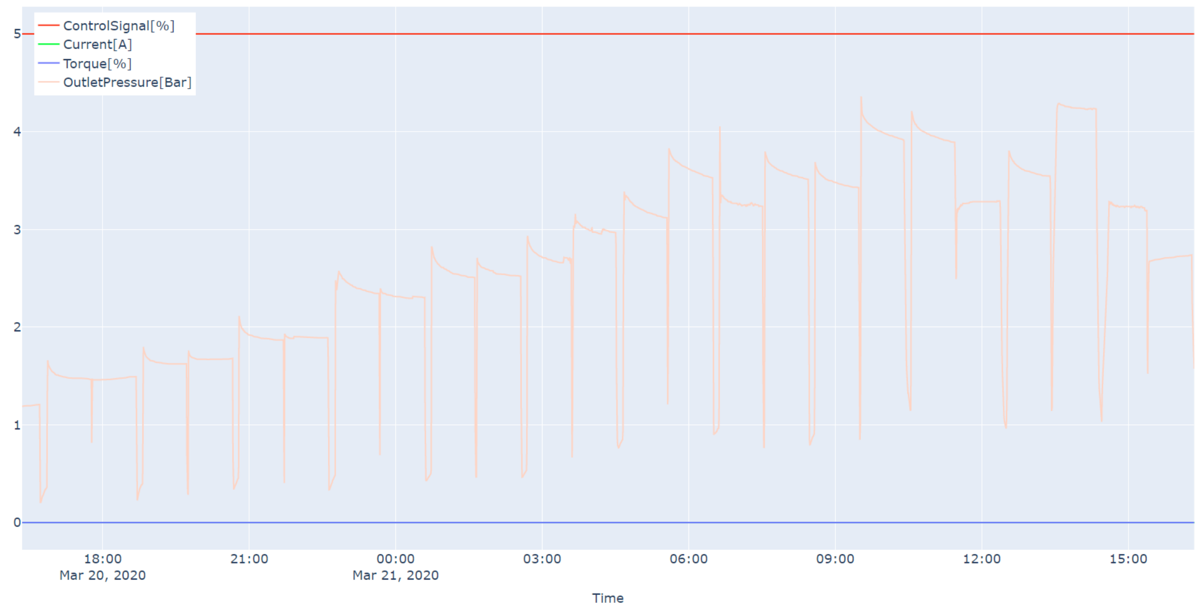


Figure 4.5 Unscaled pump 14 data (HYG_PU14)

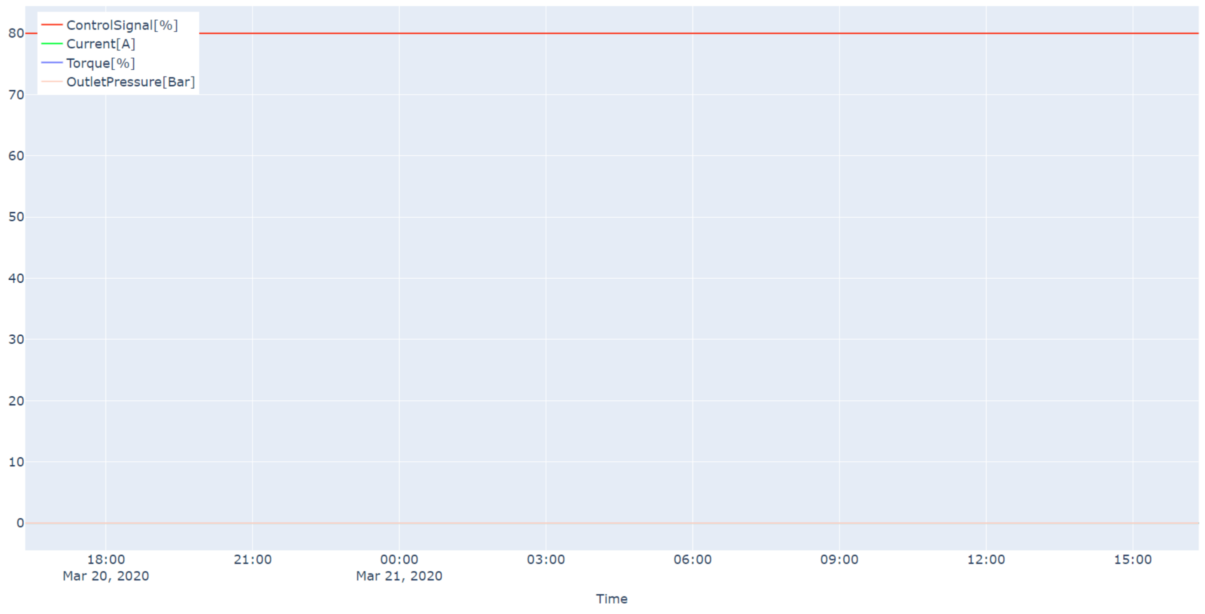


Figure 4.6 Unscaled pump 18 data (HYG_PU18)

An interesting note is that the current and thus also the torque increases in variance as the control signal increases, this can be seen in Figure 4.7. It may indicate that the standard deviation of the signals may be useful to the model.

When the scaler is applied to the data, the pressure variations are clearer in Figure 4.8, but does not seem to be dependent on any of the other measurements. This may be due to there being a pump in series after Pump 15 that sucks to control its inlet pressure to 1 bar.

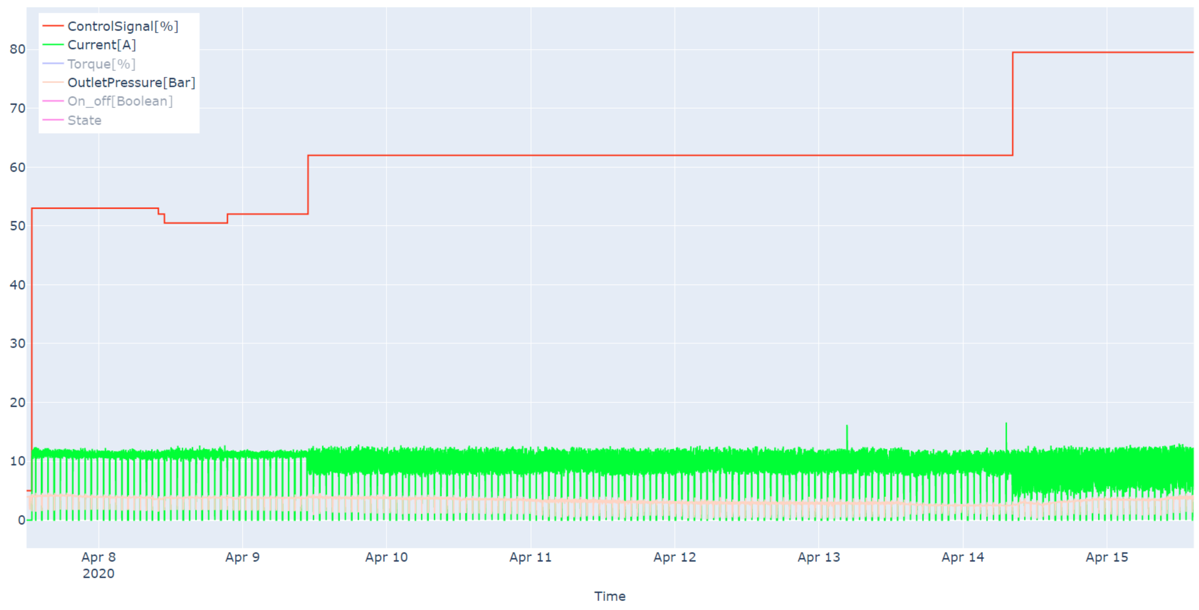


Figure 4.7 Changes in control signal to Pump 15 (HYG_PU15) increases the variation in current drastically, while pressure seems rather independent of the control signal. The speed seems to increase linearly with the control signal.

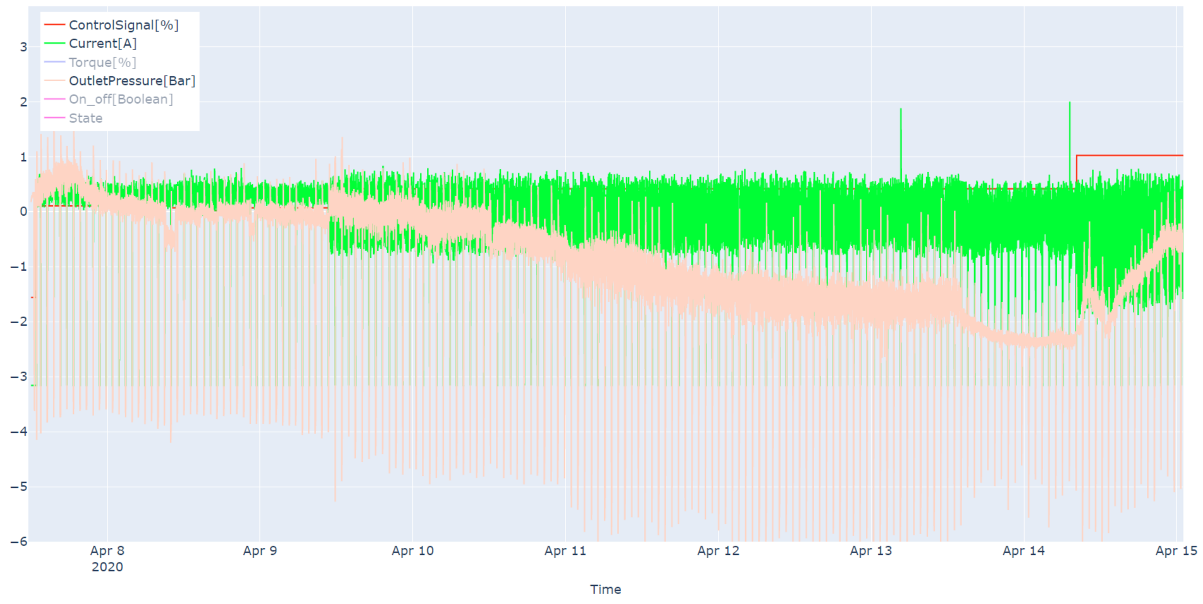


Figure 4.8 Scaled version of Figure 4.7

It should also be noted that not all the pumps are controlled by a PI controller, some are set based on other criteria. Pumps 12, 15 and 17 are manually controlled based on the wanted flow. Pumps 13, 14, 16, 18 and 19 are controlled by a PI controller based on the inlet pressure. This changes the behavior of the pumps, yet the relationships are assumed to endure. As with Pump 15, it can be seen in Figure 4.9 that the current in green also increases in variation as the control signal in red increases for Pump 16 which is PI controlled. Also, the outlet pressure in beige also appears independent of the control signal and current as was seen for pump 15.

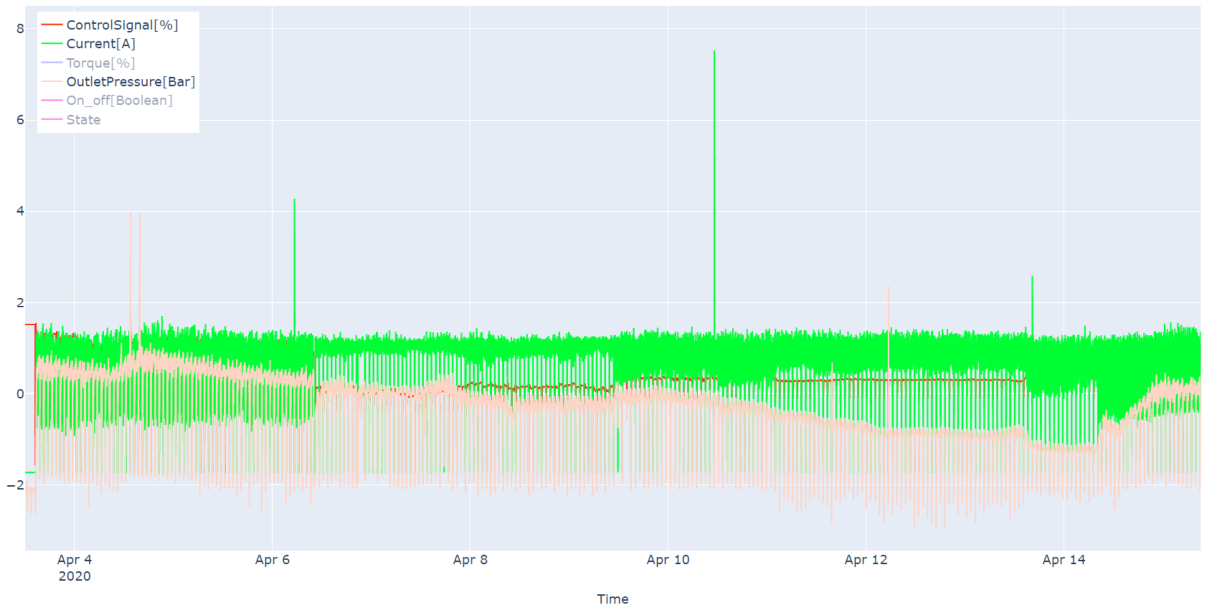


Figure 4.9 Pump 16: As was observed in pump 15, the current increases in variation as the control signal is increased. The outlet pressure again seems to be independent of the control signal

As have already been seen, some pumps are not controlled by a PI controller, this leads to the possibility that the control signal is static in the scaler fitting data. The histograms in Figure 4.10 shows the distribution of variables after scaling from pump 17 between 25. March 2021 and 4. July 2021. The control signal now lies between -70 and + 8, while the remaining variables mostly lie around 0. This can be explained by looking at Figure 4.11 and Figure 4.12. Figure 4.11 shows the control signal and speed after scaling, while Figure 4.12 shows the same variables before scaling. The control signal simply didn't have any change in the scaler calibration period and the variance of the scaler is thus 0. Essentially, what the scaler does is to move the control signal from 70 to zero, that is centering the data.

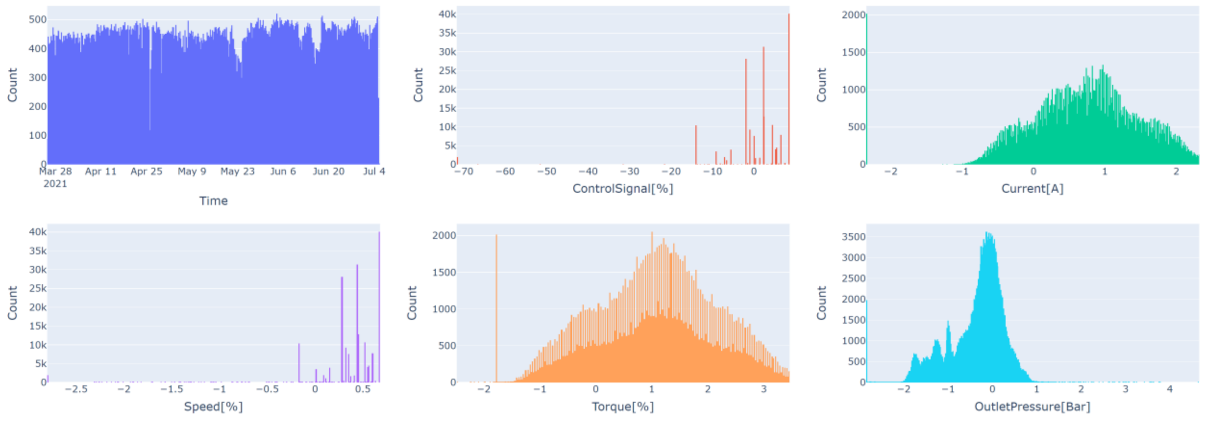


Figure 4.10 Scaled histograms from pump 17 without PI controller

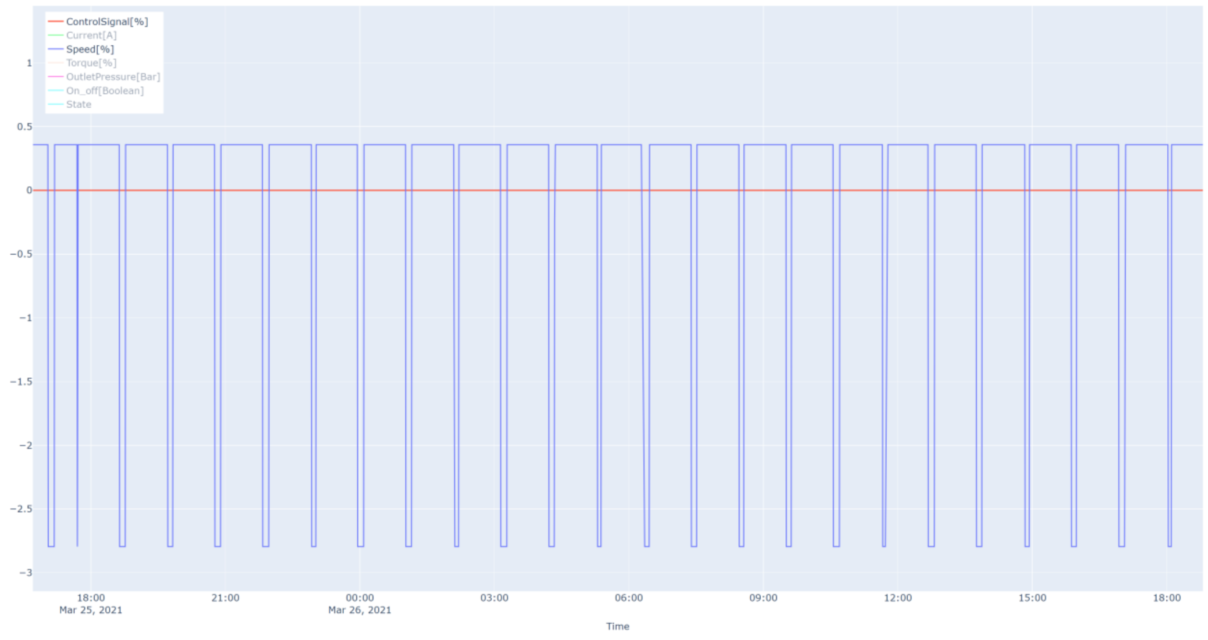


Figure 4.11 Scaled timeseries plot from pump 17 where control signal in red and speed in blue is shown.

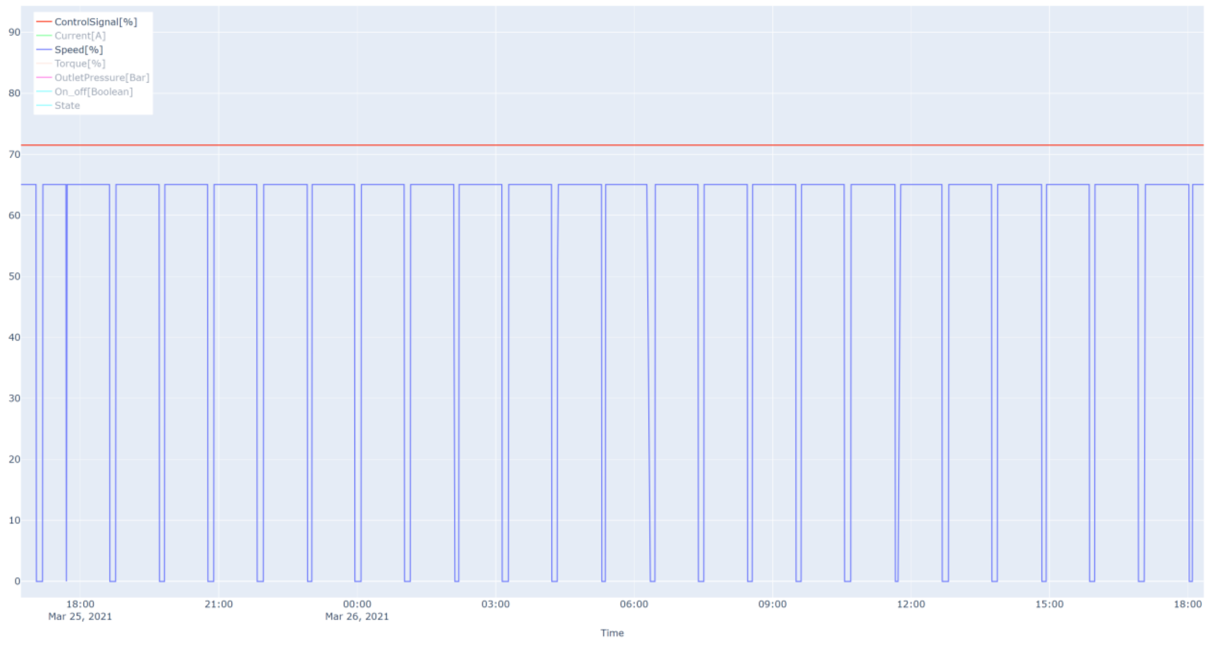


Figure 4.12 Not scaled timeseries plot from pump 17 where control signal in red and speed in blue is shown.

Figure 4.13 shows the variance of the control signal for each of the pump failures in the data. It shows the difference in variance of the data used to create the scalers. A rather large span can be seen, the span is of no issue as this is what the scaler is there for, to standardize the variation. The problem however is the ones that have 0 variance. As shown in Figure 4.10 before, this will cause the control signal to only be centred instead of scaling the data. To remedy this, the scaler data for the control signal is replaced with a linear interpolation between the bounds of the control signal, zero and 100. This should properly scale the data to be in the vicinity of zero as the other variables. This is shown in Figure 4.14 where a new scaler has been created for the data set. The control signal is now between -2 and 1, while the rest of the variables remain the same.

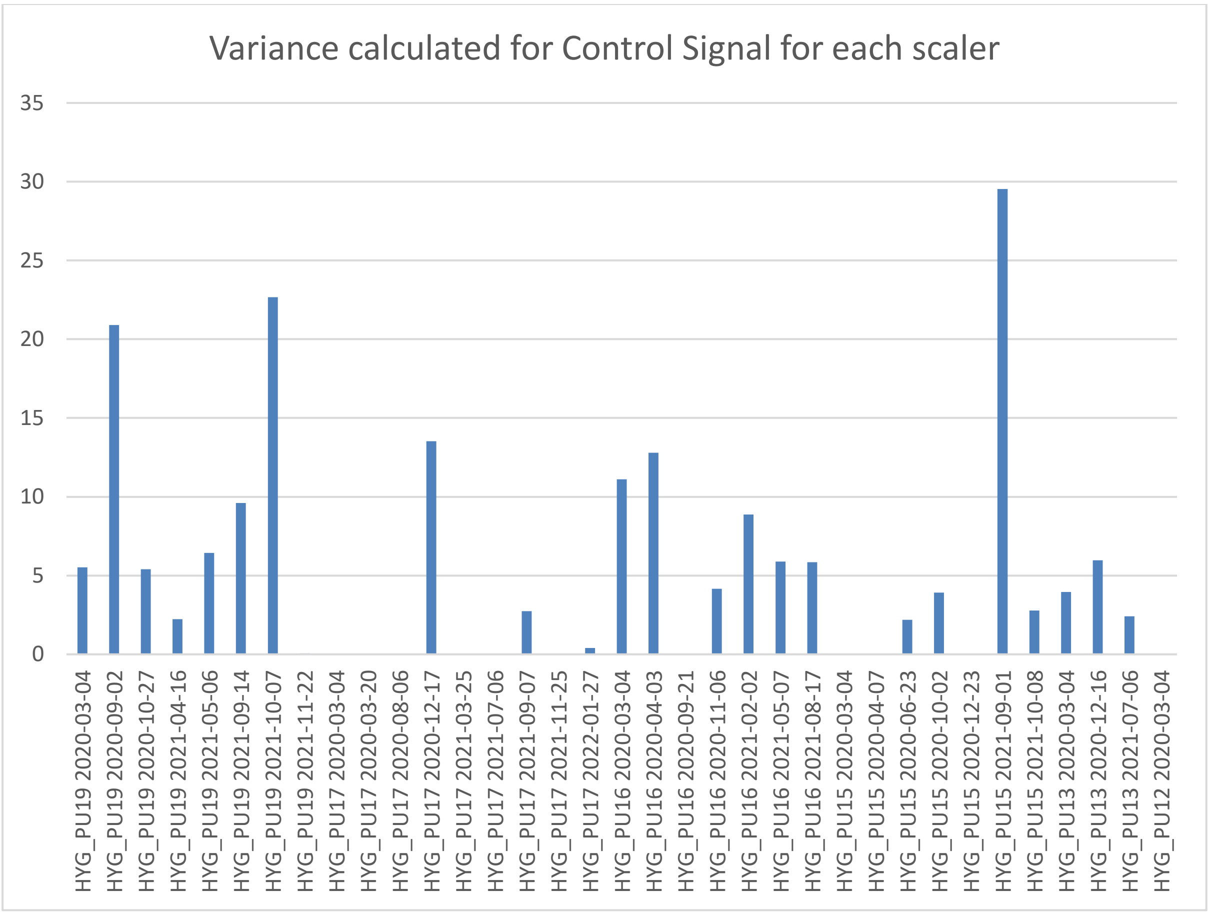


Figure 4.13 Variance of control signal from all the scaler successfully calculated

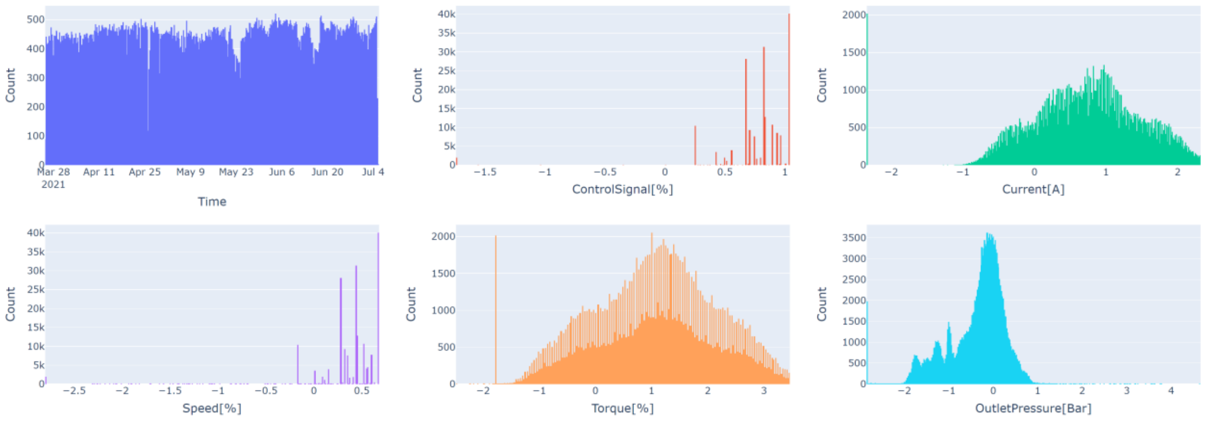


Figure 4.14 New plot of Figure 4.10 where the control signal has been rescaled

As far as it is known by the author, there have primarily been two types of pumps in these positions, with one exception, 14. September 2021 where a third type was tested. Most have been of type 1, type 2 has been tested out for some time on the most exposed positions, HYG_PU17 and HYG_PU19. Table 4.3 shows what type of pump has been installed on what position. The documentation on this is sparse, and there may be errors, however this should be better than assuming all are equal.

Table 4.3 Type of pumps installed on pump location HYG_PU17 and HYG_PU19

Date	Pump 17	Pump 19
January 2020 (04.03.2020)		Type 2
20 mars 2020	Type 1	
06 august 2020	Type 1	
02 September 2020		Type 1
27. October 2020		Type 2
17. December 2020	Type 1	
25. March 2021	Type 1	
16. April 2021		Type 1
30. April 2021		Type 1
06. May 2021		Type 1
06. July 2021	Type 1	
07. September 2021	Type 2	
14. September 2021		Type 3
07. October 2021		Type 2
22. November 2021		Type 2
25. November 2021	Type 2	
27. January 2022	Type 2	
08. February 2022		Type 2

The previous project primarily looked on the data with an assumption that each physical pump was the same and thus investigated them as one data set. Here an attempt to dig deeper into each physical pump is made to see if there are connections that are not visible when all of them is seen at the same time.

Pump 19 has eleven usable failures in the data period, Figure 4.15 shows how the data evolved over time in the leftmost column for one of the eleven pumps. The dark blue in the figure illustrates when the pump is considered ok. The red indicates that there is one week until failure, and the yellow indicates that there is less than one day remaining. The diagonal from upper left to lower right shows a scatter plot where the same variable is put up against itself. Thus, these are always straight lines.

An interesting feature here is that the control signal and speed also appear to be close to such a straight line. This indicates that the correlation is high, and that feeding both to a machine learning model may not yield any new information compared to only giving one. Similarly, the subplot between current and torque also has a high degree of correlation, it is not as clear cut as the control signal and speed.

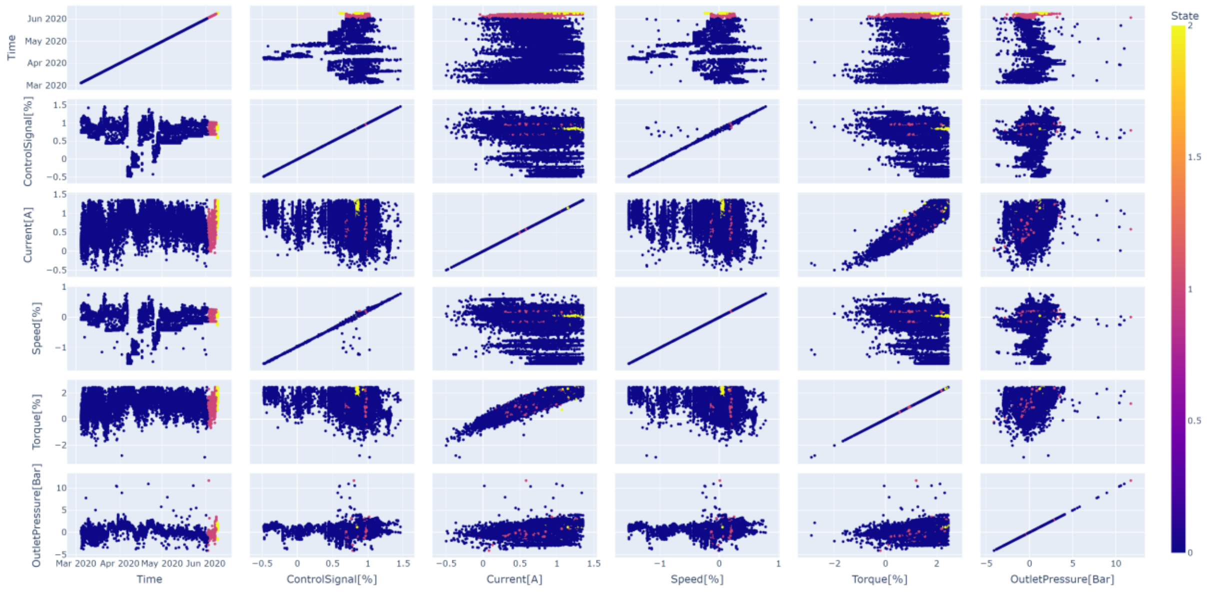


Figure 4.15 Scatter matrix plot of PU19 Type 2 pump

Figure 4.15 shows mostly the dark blue dots, *normal running*. In Figure 4.16 the same plot is shown where the last week has been highlighted. This doesn't show any clear indication that the pump is failing, rather it would seem that the pump is operating in a way it also could operate during *normal running* when looking at any two variables at the same time. It may be that certain combinations of the values will indicate the state of degradation.

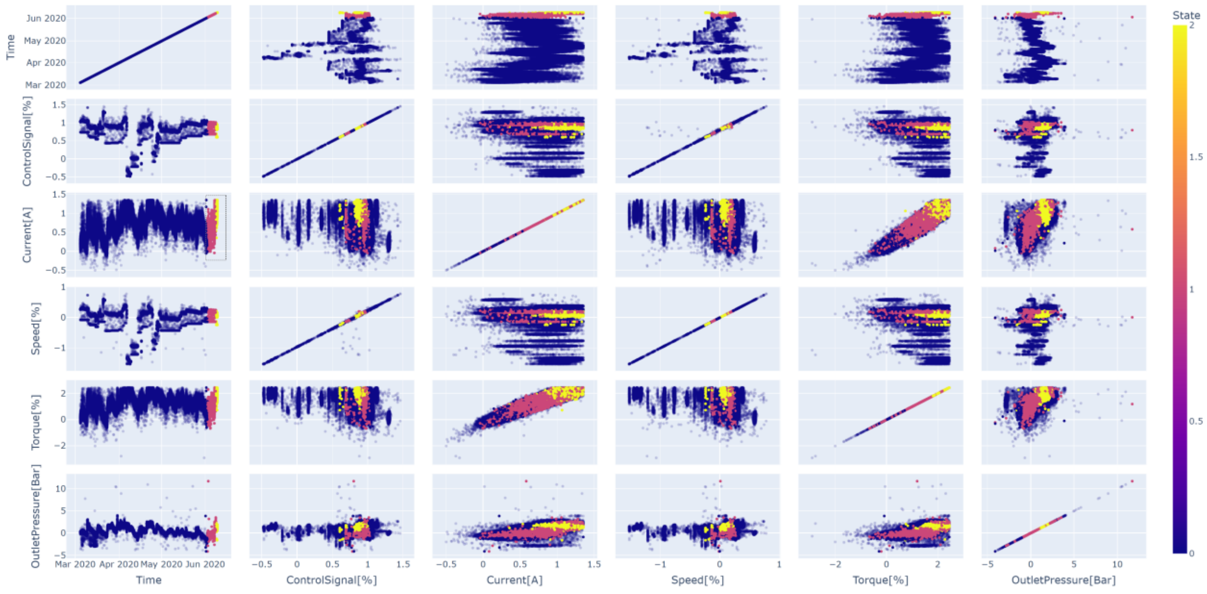


Figure 4.16 Scatter matrix plot of PU19 where *one week from failure* (red) and *one day from failure* (yellow) is highlighted. Pump type 2.

Figure 4.17 shows the lifetime of a type 2 pump which lasted very long. The control signal at first seems to increase, but then is reduced and varied, before taking on a slow increase. Meanwhile the current seems to have a steady increase over the whole lifetime. This is rather different from the previous plots where the current has been locally varying but mostly staying the same globally. These are both of type 2 which indicates that there must be some other factor, this may very well be the type of fault for the pump. Other than this, the plots look rather similar, the control signal and speed correlate and so does the current and torque. The outlet pressure has a rather high span considering that it has already been scaled and can be shown to have a similar span compared to before being scaled as seen in Figure 4.18.

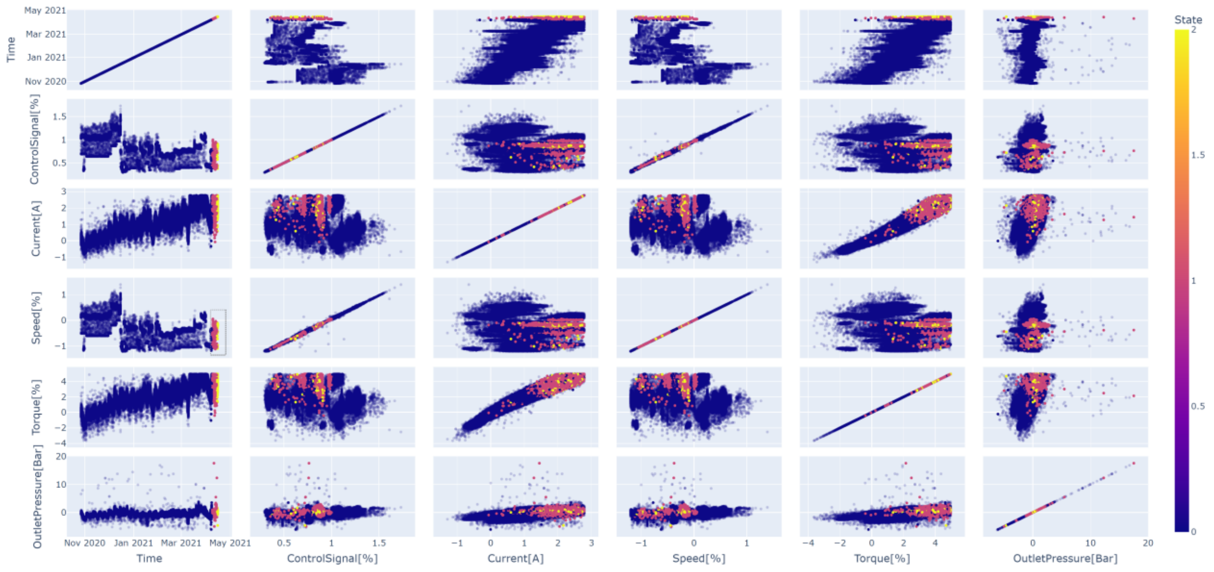


Figure 4.17 Scatter matrix plot of another pump 19 failure of pump type 2. Steady increase in current may be a feature of general degradation.

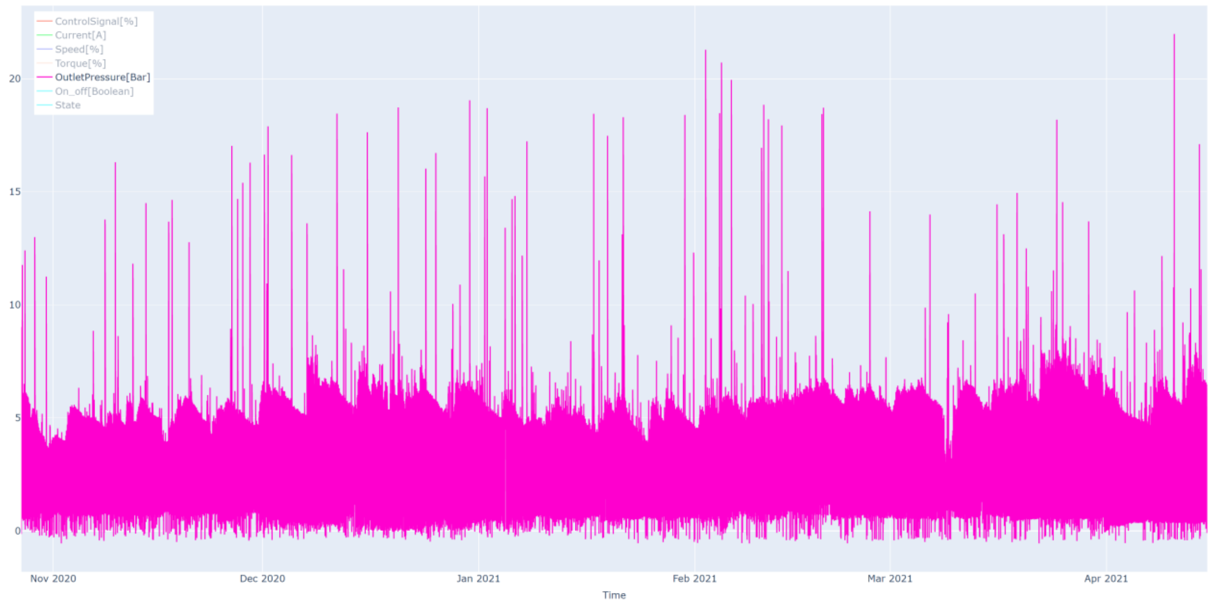


Figure 4.18 Unscaled outlet pressure from same time frame and pump as Figure 4.17

Figure 4.19 shows a type one pump in position Pump 19. As in Figure 4.17 the current and torque seem to have an increase, although the change in slope is appears lower. It can also be noticed that the plot between current and torque seems to have more variation compared to that of the type two pump.

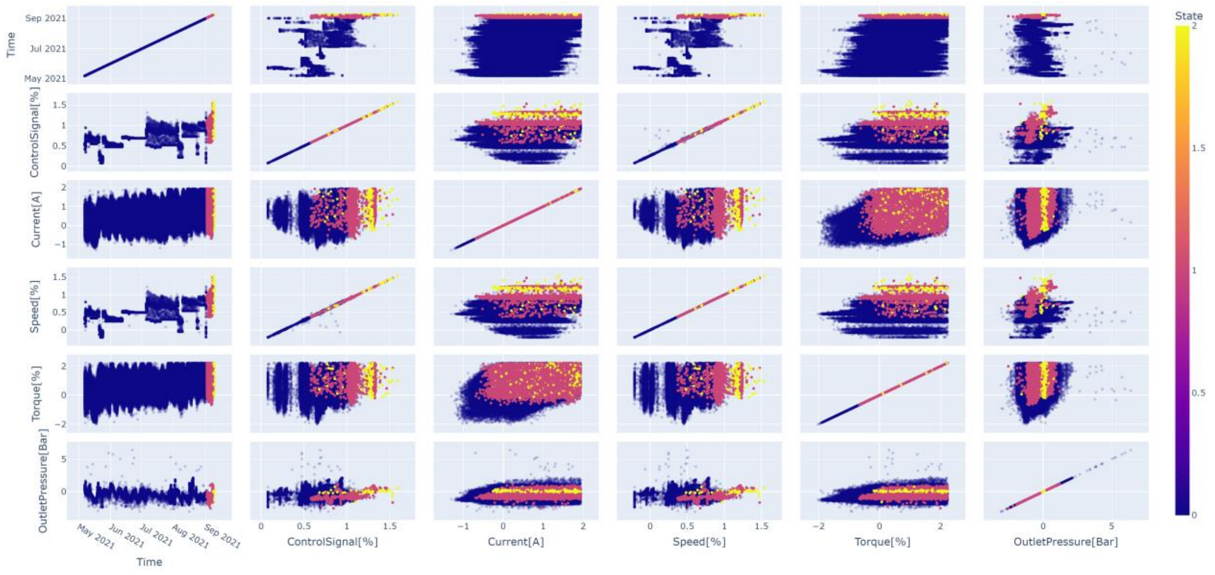


Figure 4.19 Scatter matrix plot of pump 19 failure, pump type 1

Figure 4.20 shows one failure from pump 17, it is rather different in the control signal compared to the previous plots from pump 19 as this one is not controlled by a PI controller. The current follows the trend that was observed in Figure 4.17 where it increases over time. However, this also happens to the control signal which may be to keep up the pressure or flow. It can be noted that towards the end, the outlet pressure seems to vary less and become lower. As in Figure 4.19 the scatter plot between current and torque seems to be much wider compared to the narrower pump type two and models for separate pumps may be of interest.

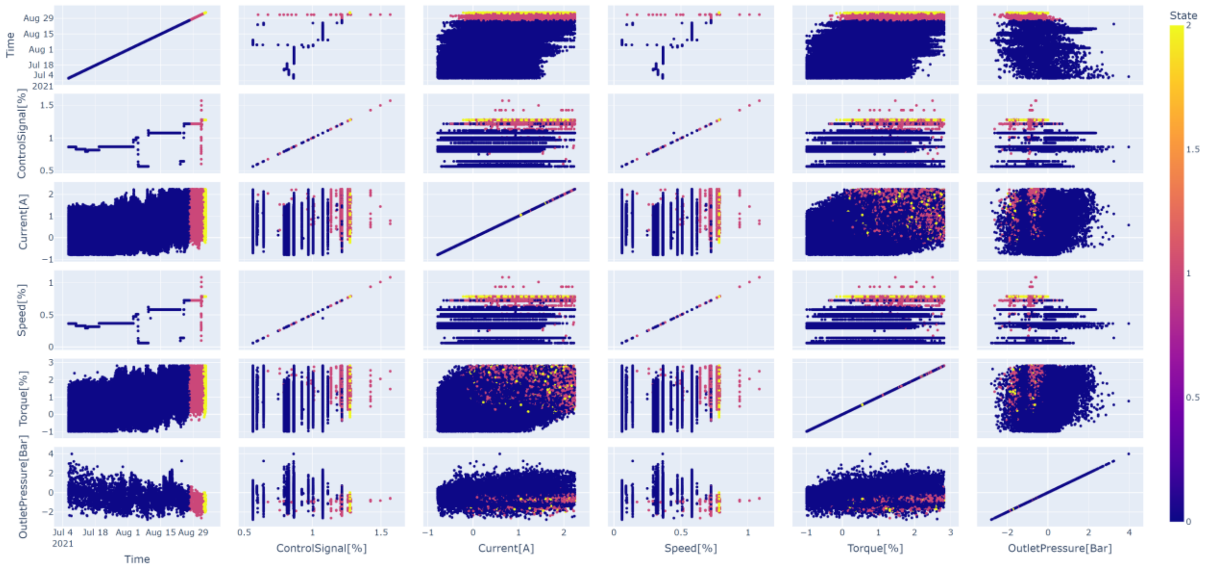


Figure 4.20 Scatter matrix plot of pump 17 type 1 without PI controller. Current and control signal is increasing as the pressure is dropping.

Figure 4.21 shows a histogram of the number of samples taken over close to 1.5 years. Some gaps can be seen where there are zero samples. This has happened several times where a pump has been replaced, that leaves only the *normal* data for training. As there is a lot of data where the pumps are *normal*, the pumps where there is no process data for the end has been omitted from training.

The pumps of this category can be seen in Table 4.4 along with other exceptions. The backup pumps have been omitted as well as these will be having samples from pressure even if they are not in use, and thus likely will be misleading for the model. It is also hard to determine when they have been replaced from operational data and the logs seldom mention them. For them to be useful, additional logic would be required.

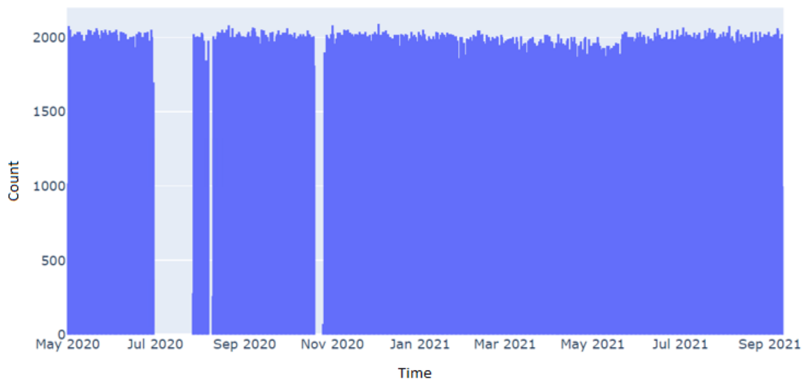


Figure 4.21 Timeseries count of measurements, gap in count indicate that there are no measurements

Table 4.4 Pumps that has been omitted from training

Tag	Pump number	Start date	End date	Comment
HYG_PU17	7	25.11.2021	26.01.2022	No data
HYG_PU17	1	20.03.2020	01-30.07.2020	No data
HYG_PU19	6	30.04.2021	05.05.2021	Operating failure
HYG_PU12	0	04.03.2020	07.07.2020	No data
HYG_PU14	0	04.03.2020	Unknown, still same?	Additional logic needed to use
HYG_PU18	0	04.03.2020	22.06.2020	Additional logic needed to use
HYG_PU18	1	22.06.2020	Unknown, still same?	Additional logic needed to use

4.3.3 Feature engineering

Up until now all that has been seen is in the raw data. It has been observed that the currents variation is increasing when the control signal increases. Interestingly the current doesn't seem to get higher, only lower in the short term as seen in Figure 4.7. Long term there has been signs that the current is increasing over the pumps lifetime as seen in Figure 4.20. These two observations may indicate that short term the average current may decrease when the control signal increases, and also that the average current may increase over the lifetime of the pump. From this it may be interesting to also use the average over some time to get a longer-term information into the model. I.e., get the average from the previous 24 hours into the model.

Using this information may be analogous to understanding the context of an event. Seeing that a cup of coffee is half-full may indicate that it is cold or hot, but if it was full five minutes ago it would mean someone has been drinking it and the probability it is still hot, is somewhat high. In the other event if it was half-full five minutes ago, it may be indicative that it is cold. If this is taken several steps further, the cup may have been half-full for 30minutes, and it is quite certain it is cold, and you would like to get fresh one.

Similar to the cup of coffee it may be of interest to know if the control signal, current and output pressure is temporarily being increased or decreased, or if it is increasing or decreasing over a longer period. It could also show deviations from the overall trend, if a given variable increases over time, but at some point, has a dip it may indicate that something unusual is happening. Physically this can indicate that some solid has been through the pump or some other event is happening.

On the other hand, a large change in mean may indicate that a wear part has met a critical point and the system is trying to compensate for the damage, or if the pump is simply not able to deliver what it used to.

Figure 4.22 shows the pump variables for pump 17 where the data has been averaged over 60 minutes. That is, each sample shown here is 120 samples of the raw data averaged and is done by PostgreSQL’s avg function & TimescaleDBs time_bucket function. It seems clear that the current in light green and the torque in blue generally is increasing over time even though the control signal in red for the most part is stable. Locally the current and outlet pressure to some degree correlate, however over the lifetime the current and torque increases while the outlet pressure stays more or less the same. Figure 4.23 shows the same for another physical pump which indicates that this is a trend, at least for pumps that are not controlled by a PI controller.

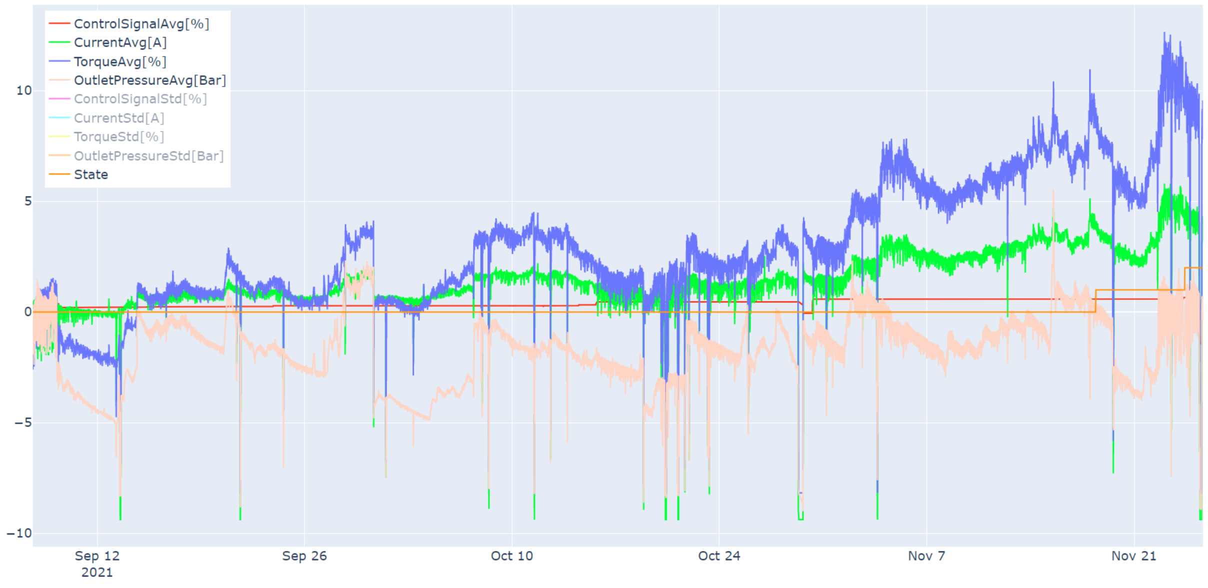


Figure 4.22 Pump 17 scaled averaged data, each sample 60minutes, over the lifetime of a physical pump

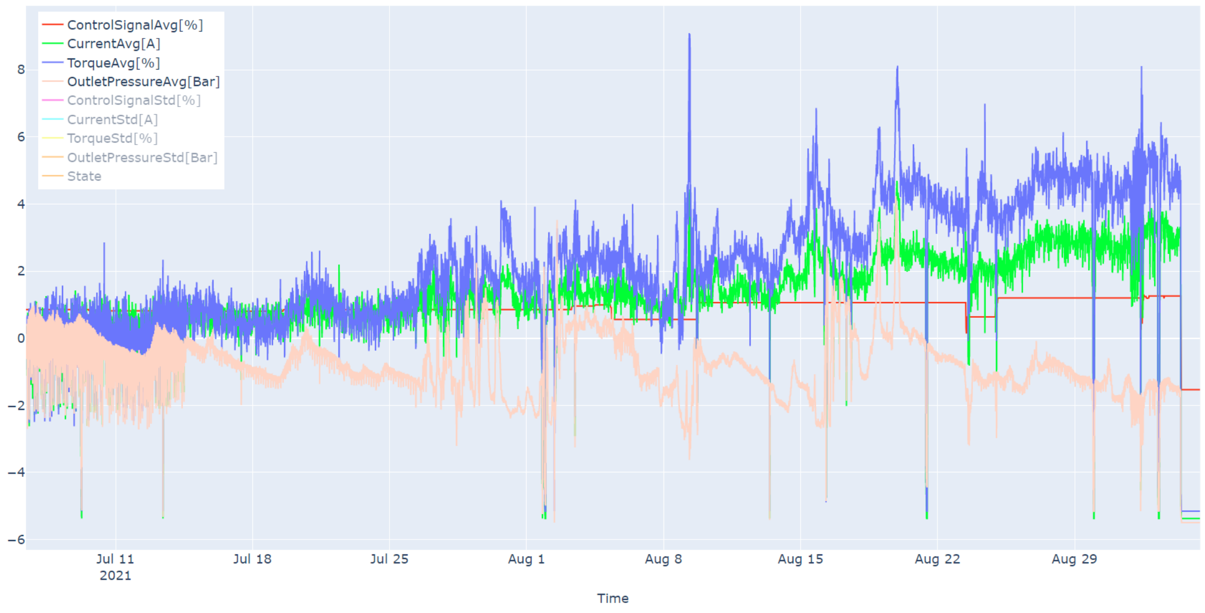


Figure 4.23 Pump 17 scaled averaged data, each sample 60minutes, over the lifetime of another physical pump

Figure 4.24 shows the standard deviation over the same time frame as Figure 4.23. Outlet pressure and current appear to have high standard deviation in the beginning before being reduced. The standard deviation goes up and down for some time, but towards the end it seems to increase for current and torque. The spikes that occur from time to time may be an indication that something is happening inside the pump.

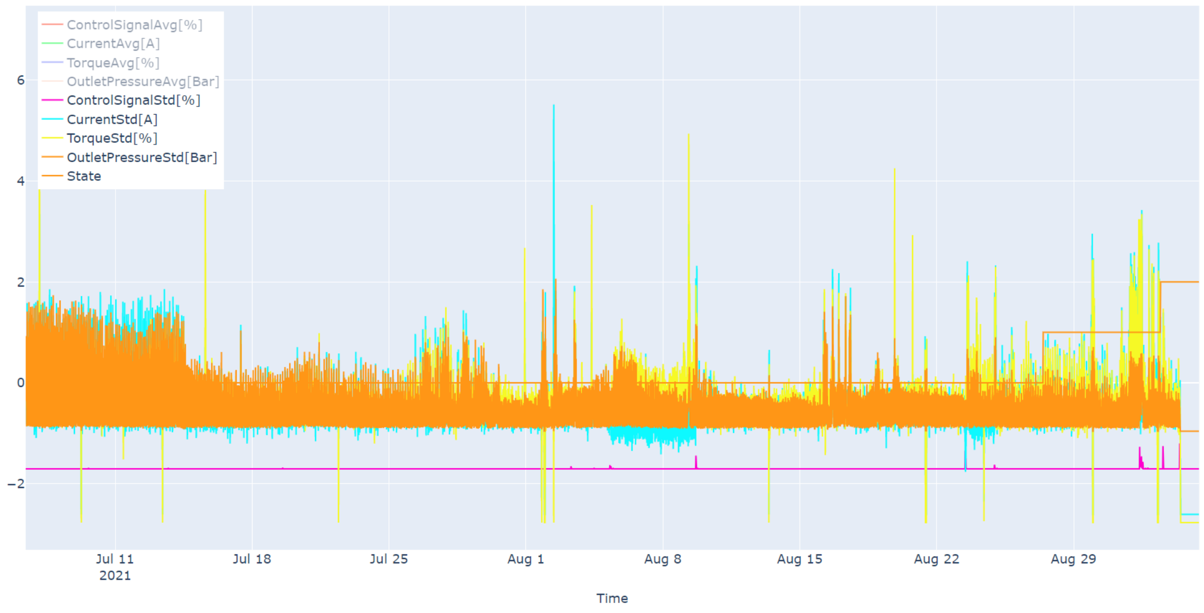


Figure 4.24 Pump 17 scaled standard deviation of data, each sample 60minutes, over the lifetime of a physical pump

4.3.4 Lurking variables

Dry matter

A lurking variable is a variable that is not included in the analysis, yet have an impact on the analysis. One such variable that is known, is the dry matter of the medium being pumped. Roughly it varies between 8-20% and is measured in a laboratory at arbitrary times. This makes it hard to add to the model as the dynamics cannot be seen from the measurements.

There are 55 samples over a two year period, although it is claimed that the dry matter various between these samples. As such there is little reason to use this variable.

Temperature

The temperature for the pumps in the system varies to a rather high degree, but there aren't good measurements for this. The inlet temperature is not measured, only the outlet from the second heat exchanger before being put into the processing tank E-11. The medium is then measured in the tank while processing, when it comes out from the tank, it is measured after the two heat exchangers.

4.4 Pre-processing

In general pre-processing is a part of machine learning that has large impact on the results of a model. This can be operations like data wrangling, transformation, aggregation, and scaling in various ways. The order of the operations may have a large impact. This subchapter will go through the pre-processing steps that will be used.

4.4.1 Narrow to wide table conversion

The data is stored in a narrow table, that is many variables are stored with a timestamp, name and value as seen in Table 4.5. This is a format that any machine learning method generally will not take. Normally a wide data structure, meaning one timestamp and one column for each variable is used.

Most data points are sampled cyclically at a rate of 30 seconds. The Boolean control signal is sampled when it changes, and the state is stored when it changes. The data is combined by using a zero order-hold join function. An example of the data before joining can be seen in Table 4.5, where the data is in one column. Here the state of pump 17 and the current of the same pump is used to illustrate. The state only has two datapoints, while the current has measurements every 30 second for close to a week. The machine learning models require the data to have one timestamp column and one column for each feature (variable). One row of data (timestamp and one value per variable) is called a sample.

Table 4.5 Measurements before joining of data

Time	Name	Measurement
11.03.2020 00:00	PU17_State	1
17.03.2020 14:31	PU17_State	2
2020-03-11 00:00:22.948	PU17_Current	15.03
2020-03-11 00:00:52.964	PU17_Current	11.66
2020-03-11 00:01:22.98	PU17_Current	9.02
…	PU17_Current	…
2020-03-17 14:29:45.956	PU17_Current	11.39
2020-03-17 14:30:15.971	PU17_Current	13.49
2020-03-17 14:30:45.987	PU17_Current	11.28
2020-03-17 14:31:15.987	PU17_Current	11.7
2020-03-17 14:31:46.002	PU17_Current	10.15
2020-03-17 14:32:16.018	PU17_Current	15.04
2020-03-17 14:32:46.034	PU17_Current	11.22

After the function has been executed the data looks like Table 4.6. The status has been forward filled until a change occurred whereupon the new value is forward filled. This of course can be used for any variables.

Table 4.6 Measurements after joining of data

Time	Name	PU17_Current	PU19_STATUS
2020-11-03 00:00:11.566	PU17_Current	15.03	1
2020-11-03 00:00:41.581	PU17_Current	11.66	1
2020-11-03 00:01:11.585	PU17_Current	9.02	1
…			
2020-03-17 14:29:45.956	PU17_Current	11.39	1
2020-03-17 14:30:15.971	PU17_Current	13.49	1
2020-03-17 14:30:45.987	PU17_Current	11.28	1
2020-03-17 14:31:15.987	PU17_Current	11.7	2
2020-03-17 14:31:46.002	PU17_Current	10.15	2
2020-03-17 14:32:16.018	PU17_Current	15.04	2

4.4.2 Sequence

LSTM takes a sequence of data and thus the features have to be split into sequences of data. The general idea can be seen in Figure 4.25, where nine timesteps of features can be seen on the upper timeline. One the sequenced timeline the nine timesteps has been split into 3 sequences of three timesteps each. These sequences are fed to the LSTM and GRU models as input vectors. The length of the sequences is somewhat arbitrary and depends on the problem. A reasonable length here, would be to cover at least one cycle of the pump, that is 120 samples. The bottom timeline shows the aggregated data fitted to the sequence, where N is an arbitrary number depending on how long the aggregation time is.

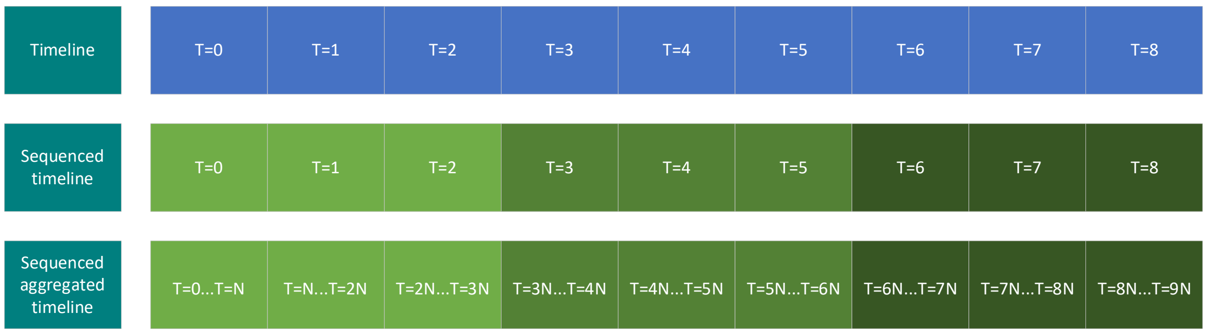


Figure 4.25 Upper timeline shows how the data is before sequencing where each blue block is the features at timestep T. The middle timeline shows the structure of the sequenced data, where each new color indicates a new sequence. The bottom timeline shows how aggregated data is fitted to the sequencing.

4.4.3 One hot encode

When attempting to classify data, the final output will be a text string. Machine learning methods does generally not accept strings as outputs, but rather demand numbers. The data is stored as a number in the database as described in Table 4.7. The loss function CategoricalCrossentropy used to train the LSTM and GRU requires the data to be in a one-hot encoded form [32]. This means each category will be one column where only one of the rows will have a 1, and the rest has a 0 as seen in Table 4.8 [33].

Table 4.7 Integer coded categories

Name	ID
Normal running	0
One week from failure	1
One day from failure	2

Table 4.8 One hot encoded categories

Normal running	One week from failure	One day from failure
1	0	0
0	1	0
0	0	1

4.4.4 Scaler

The data is scaled using standardization, equation (1) shows how to standardize a data set. In the equation, z is the resulting standardized value of a given variable, x is the input value for the variable which will be scaled, μ is the mean for the given variable data set and σ is the standard deviation of the data set.

$$z = \frac{x - \mu}{\sigma} \tag{1}$$

As was discussed in section 4.3.2, the control signal can be manually controlled by the operators, and some of the pumps are always controlled manually. This causes issues for the standard scaler, as such an evenly spaced line between 0 and 100 has been made for the control signal to be scaled by.

As in the previous project, the scalers will be created using a part of each physical pump's data. Figure 4.26 shows how this is processed. There are 8 pump positions with a few or many pump replacements, which in the figure are represented as physical pump numbers. As the positions have different tag names related to them, the tags have to be looked up depending on which pump position is being worked on. For each of the physical pumps, process measurements have to be retrieved for each of the related tags. The data is then used to create a scaler which is stored as a pickle[5] file.

The number of hours used was increased from one hour to 12 hours compared the previous project due to some starting time not being exact. The scalers then would be created with values such that some of the scaled variables became high. To mitigate this, more data was added, it is still reasonable in a practical standpoint where the model can't predict anything before 12 hours after the pump has been started for the first time.

[5] Pickle is a python library for serializing objects to file, allowing the object to be stored and read at a later point.

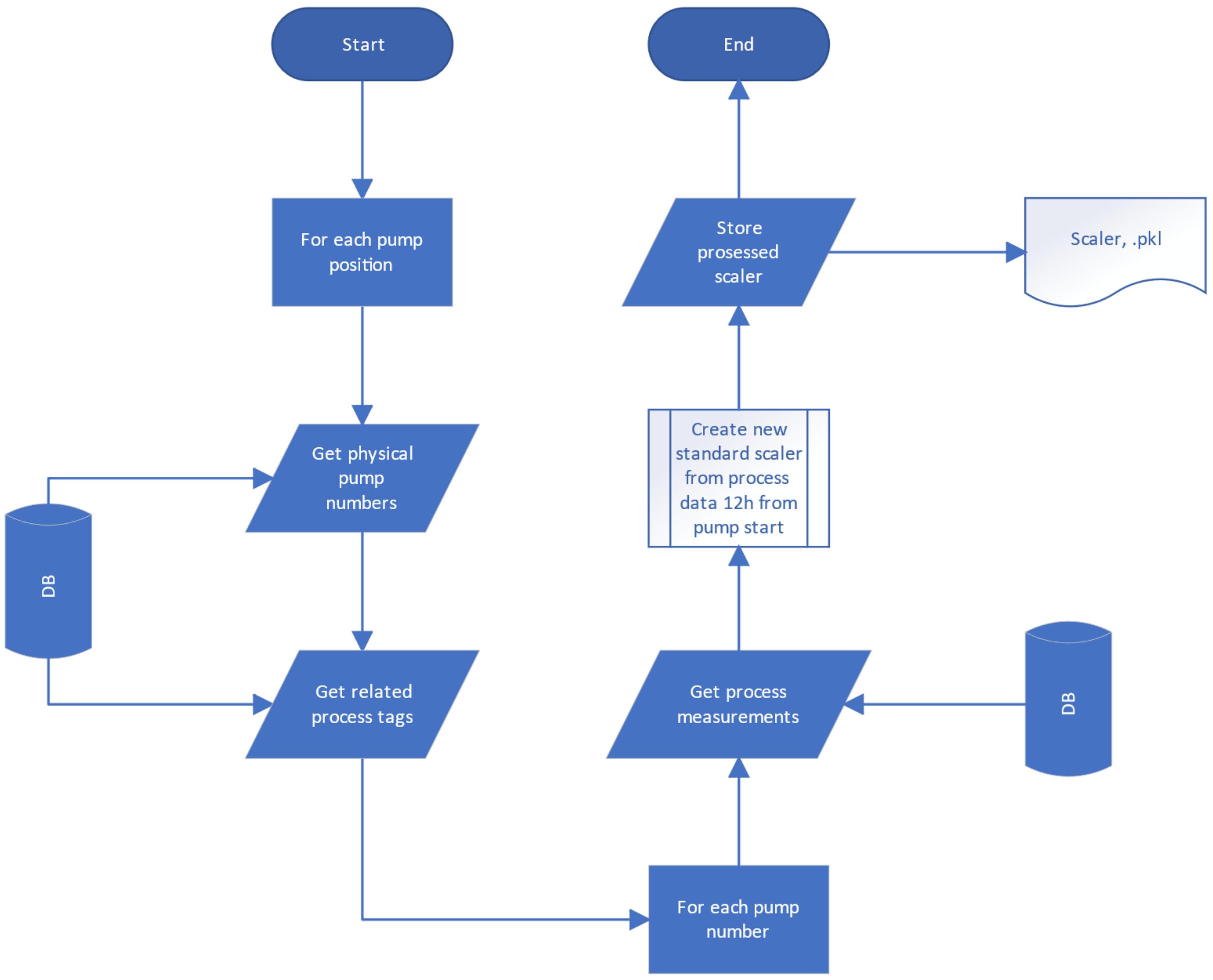


Figure 4.26 Process to create scalers for each physical pump

Figure 4.27 shows the scaled training data from all the pumps that is used for training. The control signal is expected to have some high counts as some of the pumps are controlled manually. The current has a nice distribution, although the span is high and there are a lot of spikes where the current likely has been zero before scaling. As the scalers does not have the same base due to separate scaling per pump, these zeros may vary after scaling. The torque has a somewhat wider distribution than the current, but generally follows the current. The outlet pressure also seems to be nicely distributed around zero as well, although there is still a large span.

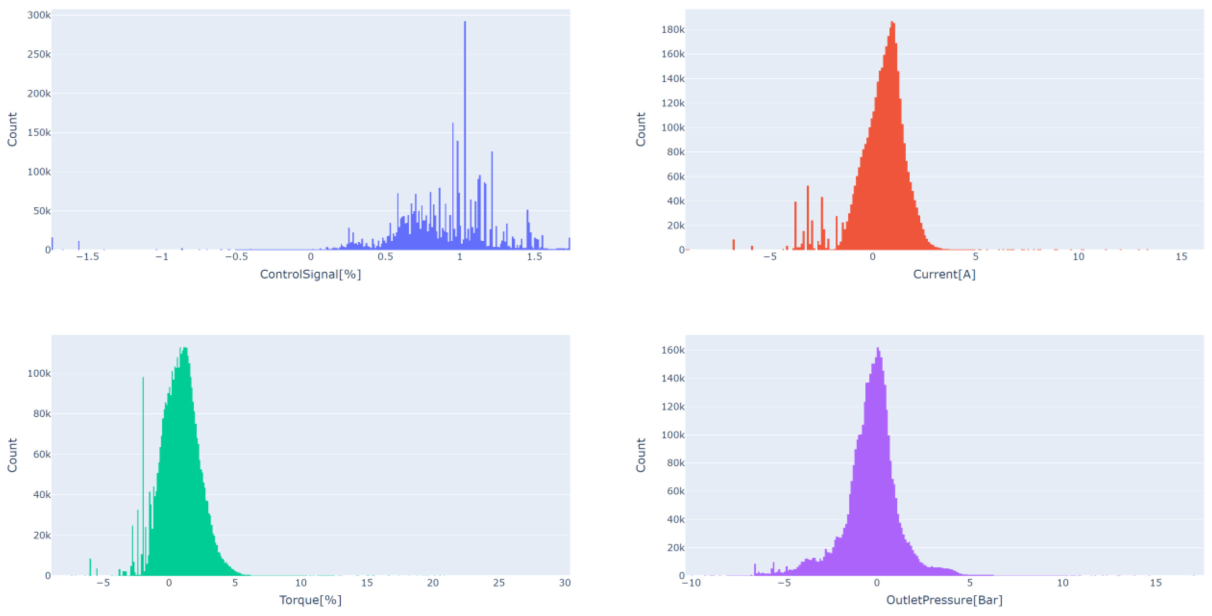


Figure 4.27 Scaled data for pumps 12, 13, 15, 16, 17 and 19

4.4.5 Outliers

As the pumps run cyclically, there are a lot of samples where the pump is off. For LSTM, these should be kept to hold the sequence intact, but for SVM this may be seen as noise. One of the easiest ways to remove these outliers is to remove the values where the pump is off by a simple filter. This is reasonable as the model does not need to make predictions when the pump is not running. In addition, there are a lot of zero values in the data set for the current and torque as these may have some time delay from start and stop signal. There is also motor starting current which gives a high current for one or two samples after starting, these can be removed by removing the highest values of the current and torque using a quantile range. The quantiles are set to keep values higher than 1% and lower than 99% of the current, torque and outlet pressure. When such values are removed, the entire sample (row of data) is removed.

Figure 4.28 shows the base features from all the training pumps after removing values where the pump is off. Most of the peaks at the lower end of the current and torque has been removed.

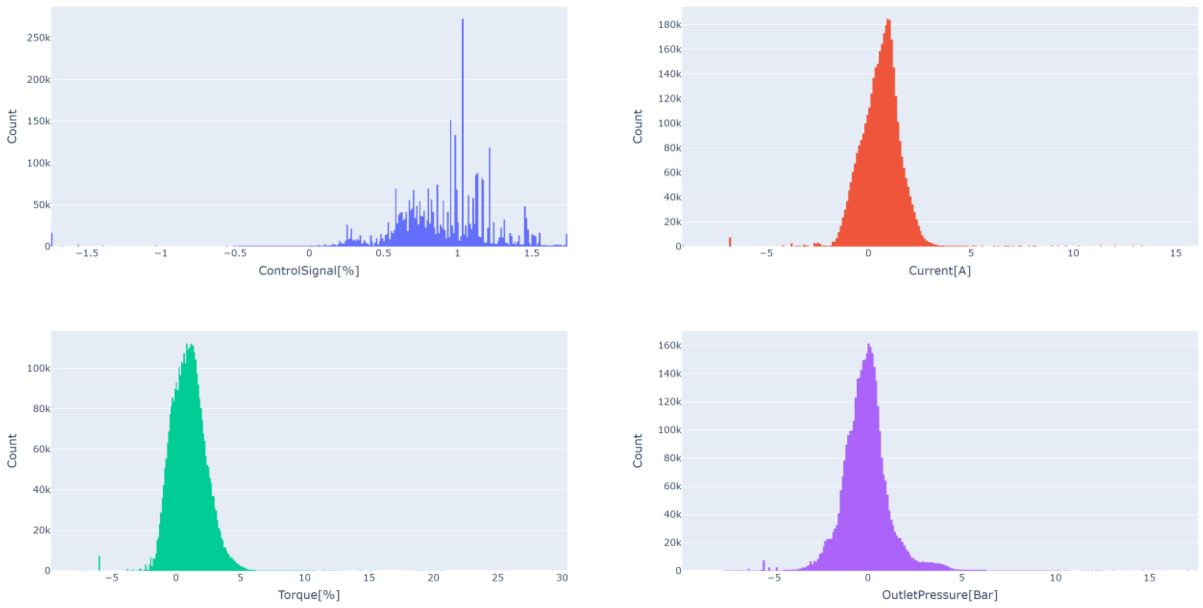


Figure 4.28 Histogram for base features after removing samples where the pump is off

Figure 4.29 shows the base features after removing the upper and lower quantiles in addition to removing samples where the pump is off. The span has been reduced greatly.

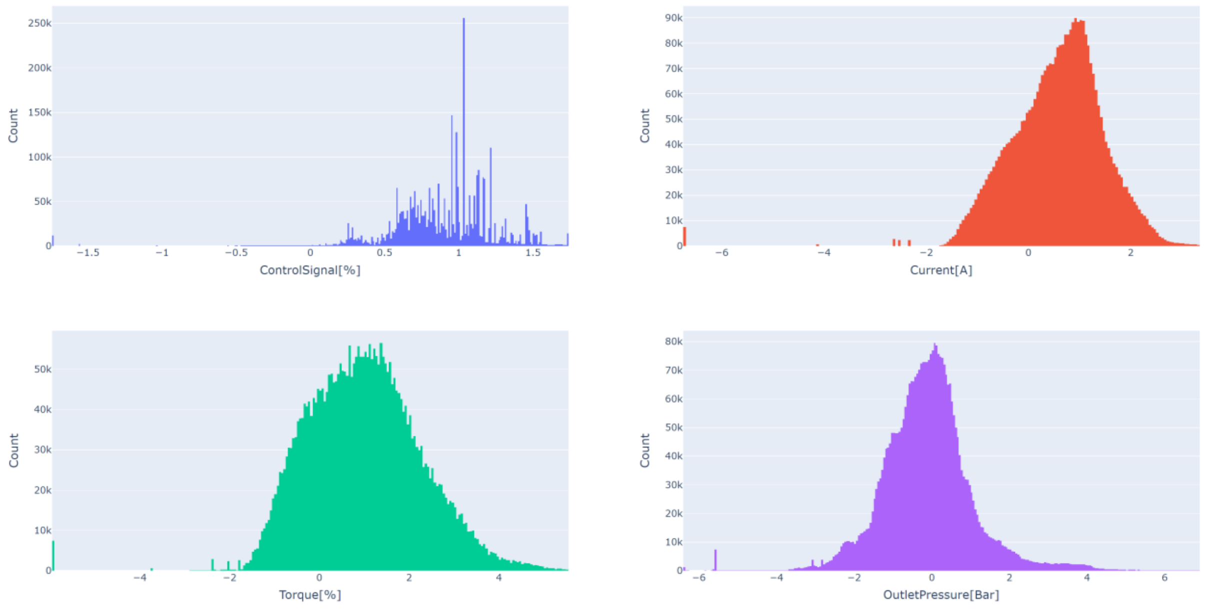


Figure 4.29 Histogram for base features after removing samples where pump is off and quantiles

4.4.6 Training, validation and test set

The data will be split in three for each pump, one training set consisting of 70%, validation set used while training of 15% and a test set of 15% which will be used when the training over all pumps is finished.

4.5 Long short-term memory

The data pipeline for the LSTM model can be seen in Figure 4.30. As for many parts of the project, the process is executed on the basis of processing one physical pump from a pump position at a time. The data for the pump number is retrieved from the database, before the data is pre-processed to fit structurally and to improve training. If it is not the first iteration of a given model, the previous checkpoint is loaded, and its weights are continued to be trained in the *train LSTM model* block. When training is finished, the weights are saved and the metadata from the training is saved to keep order of what data has been trained on. Details of the green processes will be explained further below.

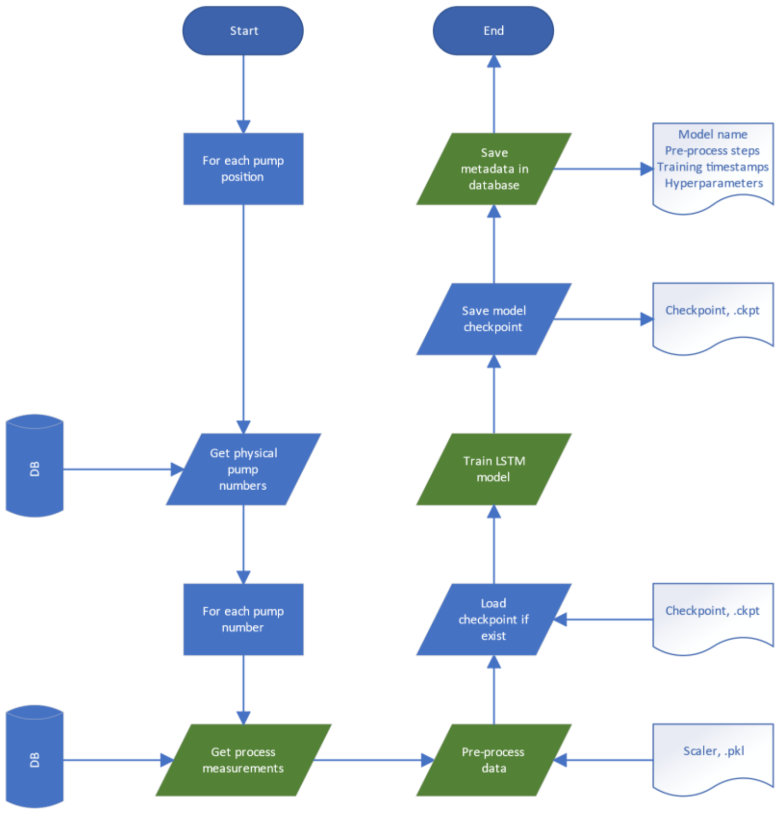


Figure 4.30 Data pipeline for LSTM and GRU models

Figure 4.31 shows what data is retrieved from the database. The amount of data retrieved is varying for each of the states. The *normal running* is the most occurring state and can last anywhere from 2-60 weeks which makes for a lot of *normal* data. To reduce training times, one day of each week where the pump is running normally is retrieved for training. For the state *one week from failure*, which lasts for six days, all the data is retrieved. The last day of the week becomes *one day from failure* and the whole last day is retrieved. These are appended before being sent to the pre-processing block. The result can be seen in a timeseries in Figure 4.32 for an illustration of how the data looks. To further describe; in time there is a gap after this function, it is however not empty between these sample times in the data fed to the model. This function has alternatives which adds aggregated data as well, but this is omitted as the underlying principle is the same except for the database query. When using only aggregates, all data is used as it already is reduced from 44-120 times for the experiments set up at the end of this subchapter.

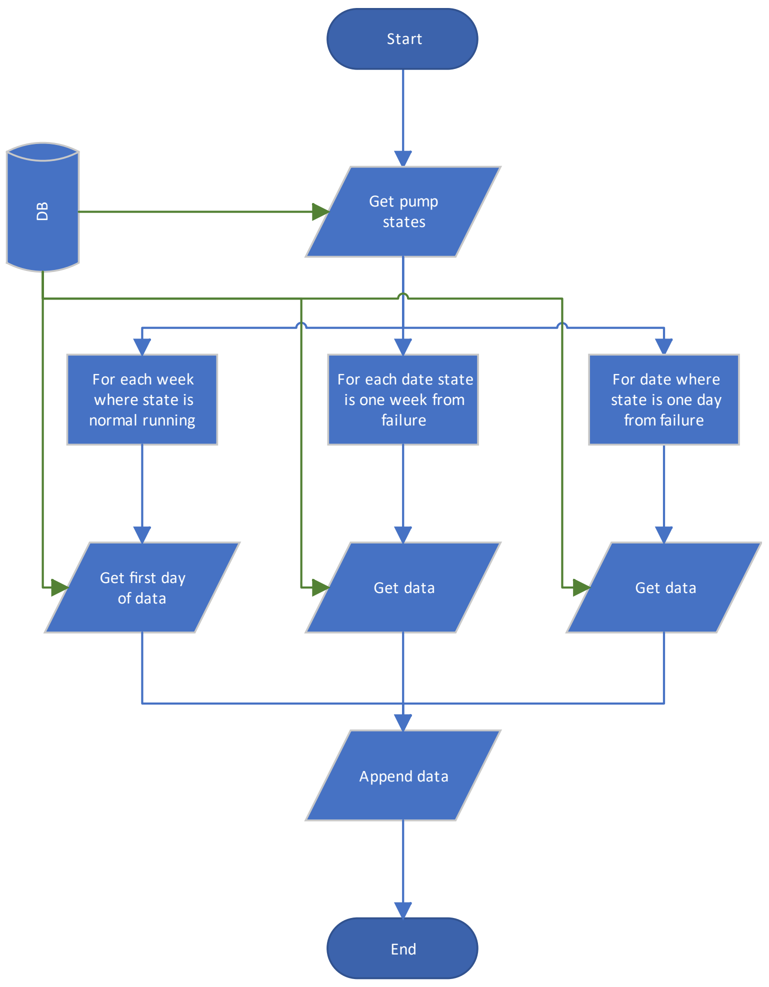

Start
DB
Get pump states
For each week where state is normal running
For each date state is one week from failure
For date where state is one day from failure
Get first day of data
Get data
Get data
Append data
End

Figure 4.31 Subprocess get process measurements

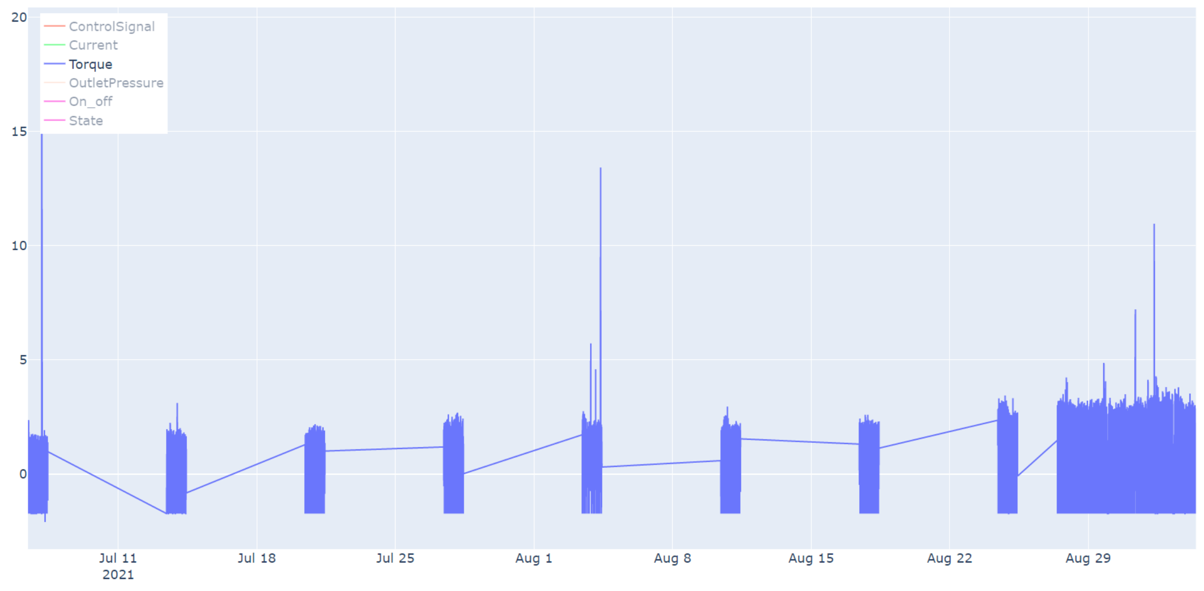


Figure 4.32 Retrieved data from process described in Figure 4.31 where one day of each week is taken where data is considered *normal*, six days where data is considered *one week from failure* and the last day of the pump before being replaced considered *one day from failure*.

The pre-processing step is shown in Figure 4.33. As explained earlier, the narrow to wide conversion is needed. The features are renamed from its original tag name to a generic name such as HYG_PU19_PW_PV become Current so that no matter which pump it comes from it’s always called Current. If there are empty or not a number (NaN[6]) cells in the data set, it should be filled first forward such as to copy the previous value, if there are cells at the start which is empty or NaN it should fill backward to use the value which would follow. The data is then split into sequences of varying lengths depending on the experiment in Table 4.9. The sequencing must happen before splitting the data into training, validation and test set as the splitting takes random samples out and would break the continuity of the sequences. Finally, the output is one hot encoded before training. After pre-processing the data, the previous model checkpoint is loaded if it exists before the training starts.

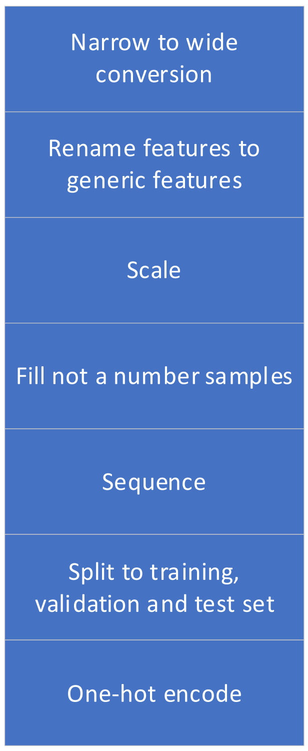

Narrow to wide conversion
Rename features to generic features
Scale
Fill not a number samples
Sequence
Split to training, validation and test set
One-hot encode

Figure 4.33 LSTM pre-process pipeline

The training experiments can be seen in Table 4.9. The number column is an identity of the experiment. The features can be either Raw, which means the features control signal (Boolean and analogue), current, torque and outlet pressure. Avg are the average of each of the mentioned features with the exception of Boolean control signal. Std is the standard deviation of the features also with the exception of the Boolean control signal. Calc is calculations between the analogue signals seen in Figure 4.34. The aggregation time is only valid if there are any aggregations such as average of standard deviation. The sequence step length indicates how long each sequence given to the LSTM model should be. Layers indicate the number of LSTM block layers are in the network. Number of neurons indicate how many neurons are in each layer in the experiment where the first number is the first layer, and the second number is the second layer. Stateful indicates whether the network will remember between sequences or if the cell state is reset in each sequence. The batch size parameter decides how many sequences will be used per update of the weights, the batch size can be used to control the training time. Increasing the batch size, reduces training times it may reduce accuracy however. An epoch tells the model how many times the data should be

[6] NaN, Not a number

iterated over. Max epochs indicates that there should be a maximum number of epochs when early stopping is used, this means that even though the max epochs might be 50, the model might stop at 15 due to the validation accuracy reducing. A parameter patience controls how many epochs the validation accuracy must be reduced before early stopping intervenes, for all models it has been set to three. The skip parameter decides which type of pumps should be skipped during training, this is related to Table 4.3.

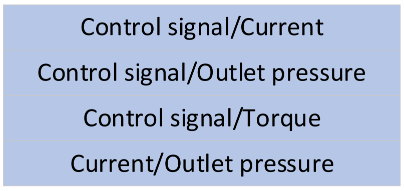

Control signal/Current
Control signal/Outlet pressure
Control signal/Torque
Current/Outlet pressure

Figure 4.34 Calculated features

Table 4.9 LSTM experiments

Number	Features	Aggregation time	Sequence steps	Layers	No of Neurons	Stateful	Batch size	Max epochs	Skip
1	Raw|Calc	0	128	1	6	No	32	50	None
2	Raw|Calc	0	128	2	6|6	No	32	50	None
3	Raw|Avg|Std|Calc	22min	128	1	12	No	32	50	None
4	Raw|Avg|Std|Calc	22min	128	2	12|6	No	32	50	None
5	Avg|Std	22min	16	2	12|6	No	32	5	None
6	Avg|Std	22min	16	2	12|6	Yes	1	5	None
7	Avg|Std	60min	16	2	12|6	No	32	5	None
8	Raw|Avg|Std	22min	128	1	4	Yes	1	5	None
9	Raw|Avg|Std	22min	128	2	6|6	No	32	5	None
10	Raw	0	120	2	32|16	No	64	5	Type 2
11	Raw|Avg|Std	22min	120	3	32|32|16	No	64	5	None
12	Avg|Std	22min	16	1	8	No	32	5	None

13	Avg|Std	22min	16	1	6	No	32	5	None
14	Avg|Std	22min	16	3	64|32|16	No	32	5	None
15	Raw	0	120	2	32|16	No	32	5	None

For each pump iteration of training, a training run number together with a model name is saved to identify the previous training such that the checkpoint can be loaded properly. Together with this information, the features, pre-process steps, when the training starts and stops and on what time frame is trained on. The train, validation, test split, layers number of neurons, checkpoint path, whether the model is stateful are stored for each pump. The test accuracy is updated after the model has trained on all training examples.

4.6 Gated recurrent unit

The gated recurrent unit uses the same data pipeline as the LSTM model, instead of LSTM blocks, there are GRU blocks in the model architecture. The experiments can be seen in Table 4.10.

Table 4.10 GRU experiments

Number	Features	Aggregation time	Sequence steps	Layers	No of Neurons	Stateful	Batch size	Max epochs	Skip
1	Raw|Calc	0	128	1	6	No	32	50	None
2	Raw|Calc	0	128	2	6|6	No	32	50	None
3	Raw|Avg|Std|Calc	22min	128	1	12	No	32	50	None
4	Raw|Avg|Std|Calc	22min	128	2	12|6	No	32	50	None
5	Avg|Std	22min	16	2	12|6	No	32	50	None
6	Avg|Std	22min	16	1	6	Yes	1	50	None
7	Avg|Std	60min	16	1	6	Yes	1	50	None
8	Raw	0	128	1	6	No	32	5	None
9	Raw|Avg|Std	22min	128	2	6|6	No	32	5	None

10	Raw|Avg |Std	22min	120	1	6	No	32	50	None
11	Raw|Avg |Std	22min	120	1	6	No	32	50	Type 2
12	Raw|Avg |Std	22min	120	1	6	No	32	50	Type 1
13	Raw	0	120	2	32|16	No	64	5	Type 2
14	Raw|Avg |Std	22min	120	2	32|16	No	64	5	Type 2
15	Raw|Avg |Std	22min	120	3	32|32| 16	No	64	5	None
16	Avg|Std	60	16	2	6|6	Yes	1	5	None

4.7 Support vector machine

The training pipeline is rather similar to that of LSTM and GRU, however Scikit-learns implementation of Support Vector Classifier (SVC) does not allow for partial fitting. Thus, all the data that will be used has to be included in one data set (and fit in memory). The same method for getting the data is used here as in LSTM and GRU, where one day of each week for all *normal* data points are used, six days of *one week from failure* and the *one day from failure*.

The pre-processing pipeline can be seen in Figure 4.36. The data must be structured in a wide-table manner, the features renamed to generalize, the data should be scaled using the created scaler, and empty samples filled, it is also split into training, validation and test set. Some outliers are removed to reduce the number of extreme values, this is particular for the current and torque, where the top and bottom 1% of data points are removed. Finally, the outputs are one-hot encoded.

Before training the model, the data for all the pumps are gathered up into one coherent data set in the *Restructure da*ta *for all pum*ps block.

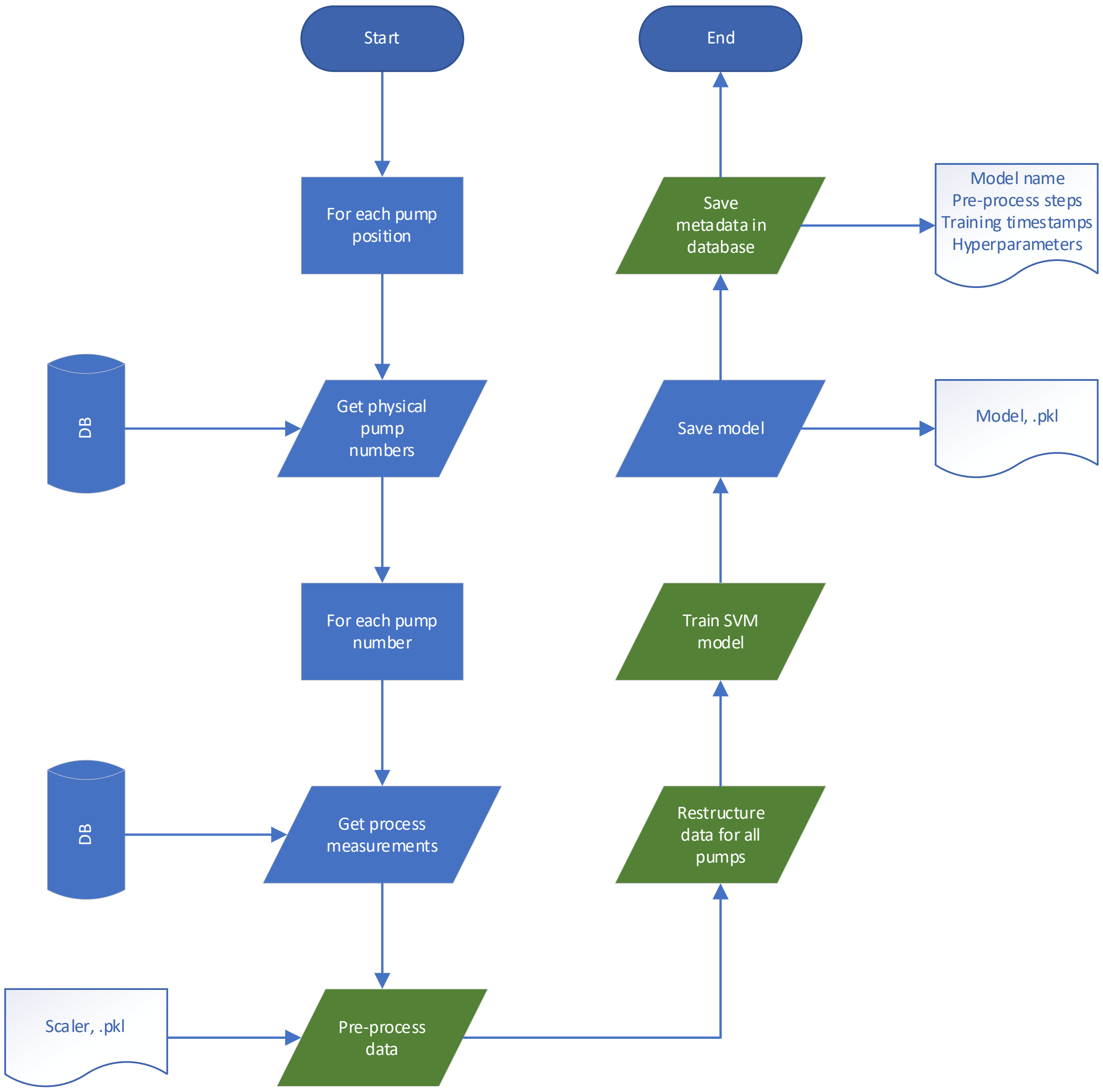


Figure 4.35 Training pipeline for Support Vector Machine

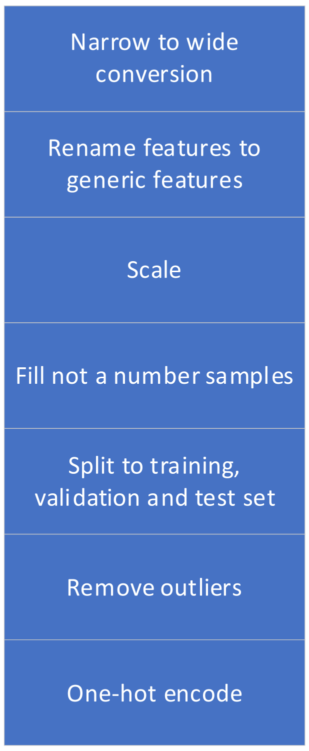

Narrow to wide conversion
Rename features to generic features
Scale
Fill not a number samples
Split to training, validation and test set
Remove outliers
One-hot encode

Figure 4.36 SVM pre-process pipeline

SVM has several parameters that may alter the results of the model. It has a kernel which can be a radial basis function, linear line, a polynomial, or a sigmoid function. Each of these four kernels has parameters which describes them. The linear kernel only has one parameter, C which is a regularization parameter, effectively deciding how much the model should be allowed to fail (and thus not overtrain). Gamma is a kernel coefficient for radial basis function, polynomial and sigmoid. Low gamma gives a high impact for each sample point, while a high gamma gives a lower impact for each sample. Low gamma tends to overtrain. For the polynomial a parameter degree decides what degree the function will be. C0ef0 is a bias term for the polynomial and sigmoid kernel function.

The outputs of the classes can be weighted in the case where there are more samples for one class than another. Scikit has a parameter for this called class weight. An option is to use a balanced argument where the SVM algorithm will find the best weights for the classes to make them even. This is set for all the SVM experiments.

Finding the hyperparameters of SVM can be tedious work and can be done using estimators from Scikit-learn [22]. These amongst others, include GridSearchCV and HalvingGridSearchCV. The SVM experiments can be seen in Table 4.11.

Table 4.11 SVM experiments

Number	Features	Kernel	C	Gamma	Degree	Coef0
1	Raw|Avg|Std|Calc	RBF	1	1		
2	Raw|Avg|Std	GridSearchCV RBF	0.1 to 1	0.001 to 0.0001		

3	Raw|Avg|Std|Calc	Linear	1			
4	Raw|Avg|Std	Halving GridSearchCV Linear	0.1 to 1			
5	Raw|Avg|Std|Calc	GridSearchCV Polynomial	0.1 to 1	0.001 to 0.0001	1 to 2	
6	Raw|Avg|Std|Calc	Polynomial	0.1	1	3	1
7	Raw|Avg|Std	Sigmoid	1	1		0
8	Raw|Avg|Std	GridSearchCV Sigmoid	0.1 to 1	0.001 to 0.0001		1
10	Raw|Avg|Std	RBF	0.1	0.1		
11	Raw|Avg|Std	RBF	0.1	10		
12	Raw|Avg|Std	RBF	0.1	50		
13	Raw|Avg|Std	RBF	0.1	100		
14	Raw|Avg|Std	RBF	1	50		
15	Raw|Avg|Std	RBF	1	5		
16	Avg|Std	RBF	1	5		
17	Avg|Std	RBF	0.5	5		
18	Avg|Std	RBF	0.5	7		

19	Avg|Std	RBF	1.5	5		
20	Raw|Avg|Std|Calc	RBF	1	Scale		

4.8 Testing

The tests are run on physical pumps which has not been used for training, the training data has been from 2020-03-01 to 2022-02-01. The tests are done as such due to the results seen in the previous project, where the models could fairly well predict the outcomes of the same pumps that was used to train with, even though the data was split into train, validation and test set. However, when tested on a new pump, the models had trouble in predicting the correct category and this can be considered a true test as it is closer to how the model would behave in production.

Some pumps started running in the training period but has not been trained on as the end of life is the most interesting part. These may have been included in the testing data if they stopped working in the period from 2022-02-01 to 2022-05-01. The pumps tested on can be seen in Table 4.12. Exceptions may occur as the data pipeline has been somewhat changed since the first model training. Thus, some models will test with higher amount of pump failures which are not listed below.

Table 4.12 Pumps used for testing models.

Date and time end of life	Tag location	Pump number
2022-01-27 16:41:00	HYG_PU19	10
2022-04-06 10:26:00	HYG_PU19	11
2022-03-30 09:30:00	HYG_PU17	8
2022-02-08 12:49:00	HYG_PU16	6
2022-04-19 22:08:00	HYG_PU13	3
2022-02-08 03:23:00	HYG_PU12	2
2022-04-18 10:04:00	HYG_PU15	6

5 Model results

This chapter will go through the results from the LSTM, GRU and SVM models, as there are many experiments done, only a handful of them will be described in detail. First the LSTM results will be presented, then the GRU results and finally the SVM results.

5.1 Long short-term memory

Many LSTM models has been trained, here some of the results will be shown in the form of confusion matrices along with network structures and details of parameters.

The first LSTM model used the raw features On/Off [Boolean], Control signal [%], Current[A], Torque[%] and Outlet Pressure [Bar]. In addition, it uses some calculated values which is considered to be various efficiency measures, these are seen in the lower part of the inputs in Figure 5.1. The inputs are connected to an LSTM block, here shown as a typical neural network layer, named Layer one in the figure with 6 neurons, meaning each of the neural networks in the LSTM block has 6 neurons. The LSTM block is then connected to the outputs of the model in a dense layer using the softmax activation function. The softmax function gives a probability distribution between the outputs, the label with the highest probability is considered the predicted value. The model is not stateful, meaning that for each sequence, the old cell state is forgotten. Each sequence is of 128 samples which equates to 64 minutes. The model was trained on pumps from tag locations PU12, 13, 15, 16, 17 and 19. This model trains on all model types described in Table 4.3.

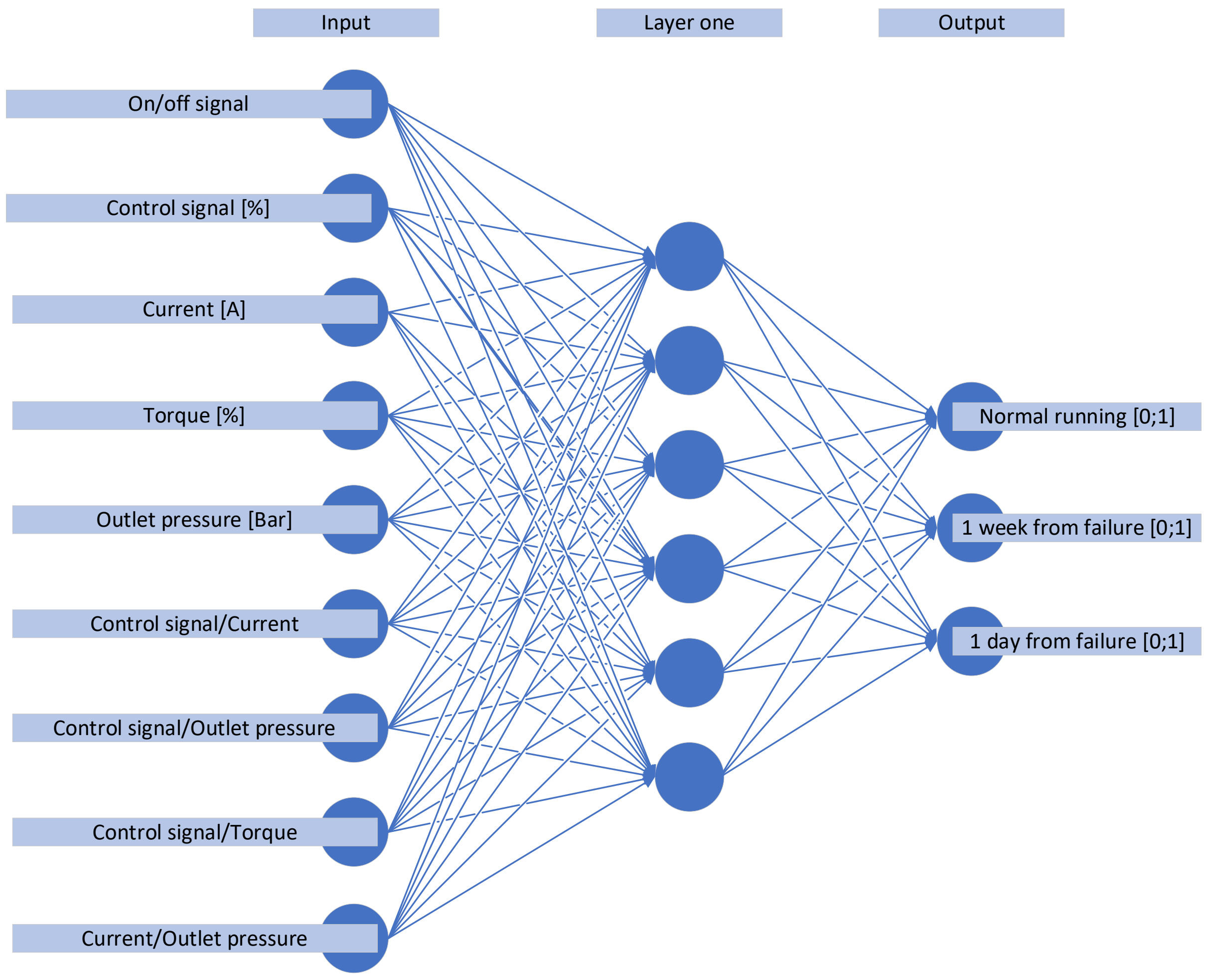


Figure 5.1 LSTM model structure for model one.

The confusion matrix in Figure 5.2 shows the results from the model. The confusion matrix shows what the true label is on the vertical axis and what the predicted label is on the horizontal. The numbers are divided by total samples over the horizontal, meaning each row adds up to one and is a decimal value between zero and one indicating how big part was predicted in each label. The diagonal from lower left to upper right shows where the model correctly predicted the labels. 100% of the *Normal* samples were correctly predicted. 17.6% of the *one week from fail* was correctly predicted, while 82.4% were predicted as *normal*. 0.2% was correctly predicted in the *one day from failure*. 28.7% of the *one day from failure* was predicted as *one week from fail* and 71.2% was predicted as *normal*.

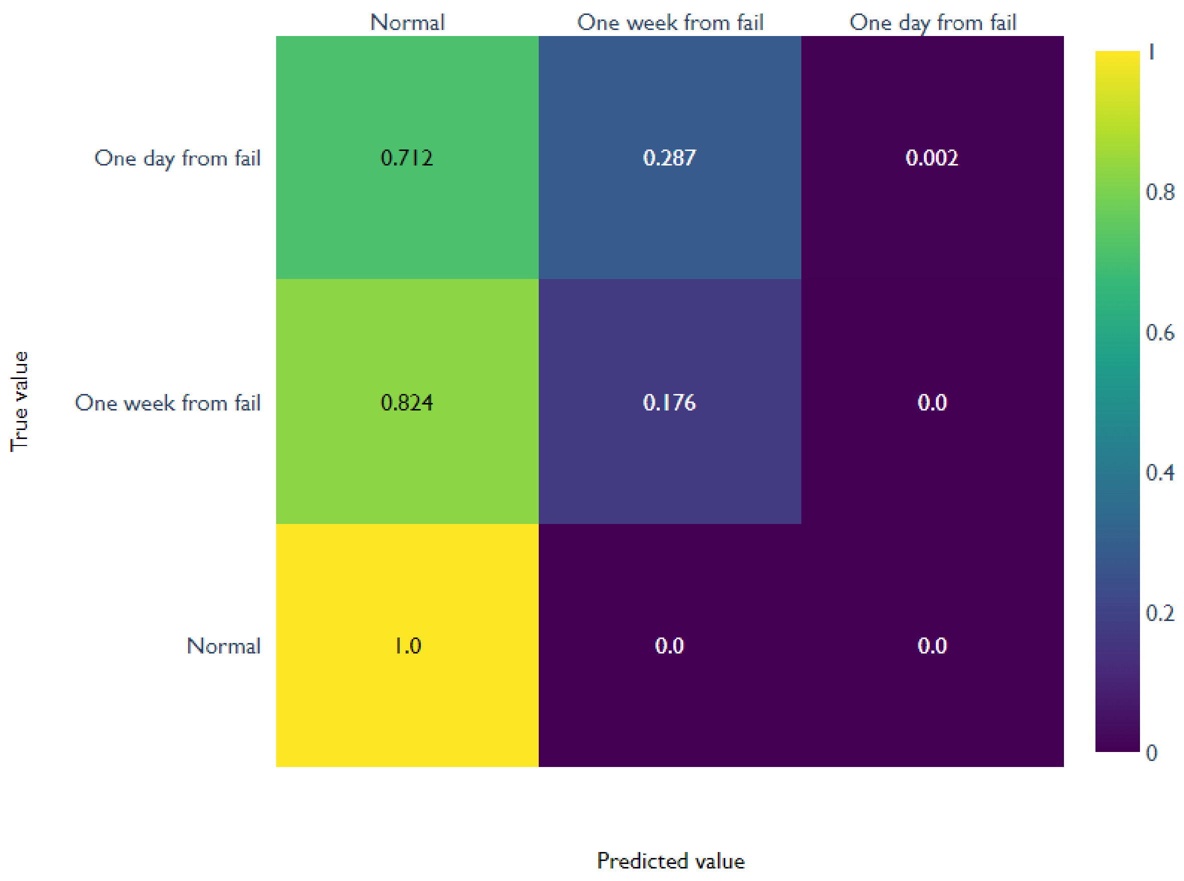


Figure 5.2 Confusion matrix for LSTM model one

LSTM model 5 used only aggregated data, the features consists of average and standard deviation of Control signal [%], Current[A], Torque[%] and Outlet Pressure [Bar] as seen in Figure 5.3. The data was aggregated over 22minutes and had a sequence length of 16, meaning each sequence looked at approximately 6 hours at the time to give an indication of the pump state. The model has two layers, the first using 12 neurons for each neural network in the LSTM block, the second using 6 neurons for each neural network in the LSTM block. The model was not stateful, so for each new sequence, earlier states are forgotten. The model was trained on all types of pumps considered in Table 4.3.

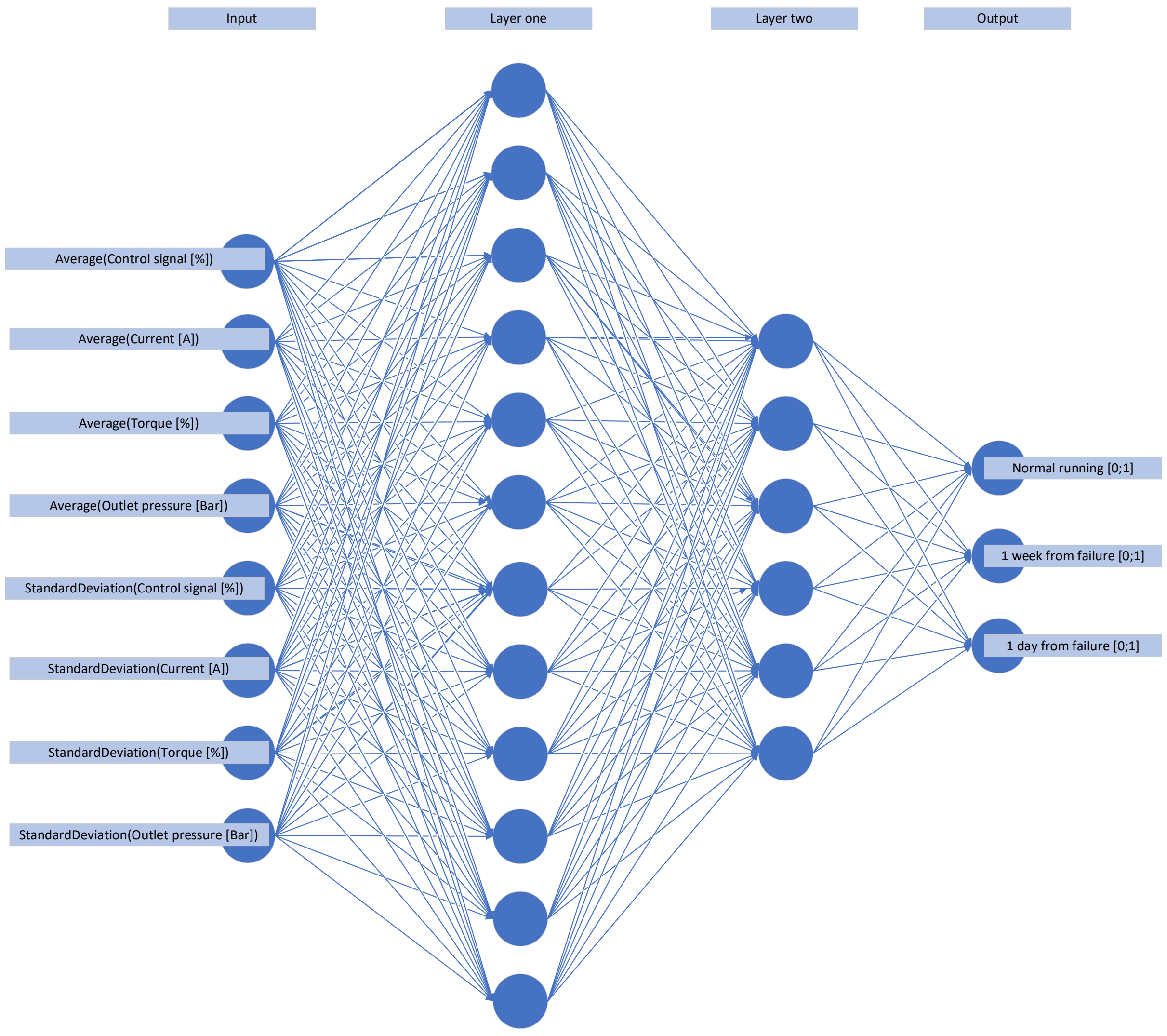

Input
Layer one
Layer two
Output
Average(Control signal [%])
Average(Current [A])
Average(Torque [%])
Average(Outlet pressure [Bar])
StandardDeviation(Control signal [%])
StandardDeviation(Current [A])
StandardDeviation(Torque [%])
StandardDeviation(Outlet pressure [Bar])
Normal running [0;1]
1 week from failure [0;1]
1 day from failure [0;1]

Figure 5.3 LSTM model 5, inputs, layers, neurons and outputs

The confusion matrix in Figure 5.4 shows the results from LSTM model 5. The confusion matrix shows what the true label is on the vertical axis and what the predicted label is on the horizontal axis. 100% of the *Normal* samples were correctly predicted. 14.7% of the *one week from fail* was correctly predicted while 85.3% were predicted as *normal*. 0% was correctly predicted in the *one day from failure*, the label was neither wrongfully predicted. 0.9% of the *one day from failure* was predicted as *one week from fail* and 99.1% was predicted as *normal*.

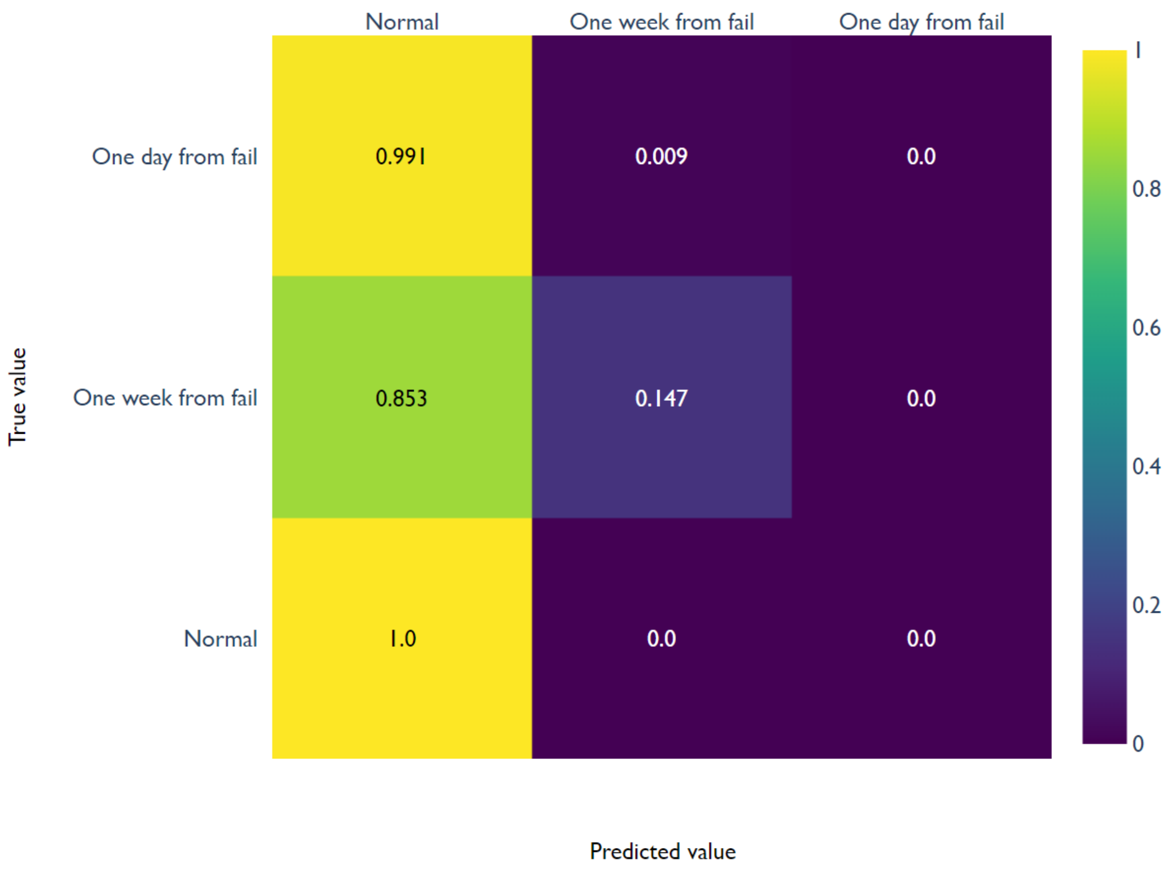


Figure 5.4 Confusion matrix from LSTM model 5, normalized correctly predicted labels are on diagonal from lower left to upper right

All of the models have, during training had a validation accuracy between 99% and 100%. This led to an idea that the model was overtraining, thus a model with fewer than recommended neurons was tested. This is model 8, it uses the raw data, average and standard deviation with 4 neurons in the LSTM layer as seen in Figure 5.5. The model was set to be stateful, meaning it will keep the cell state from the previous sequence. Each sequence was 128 samples.

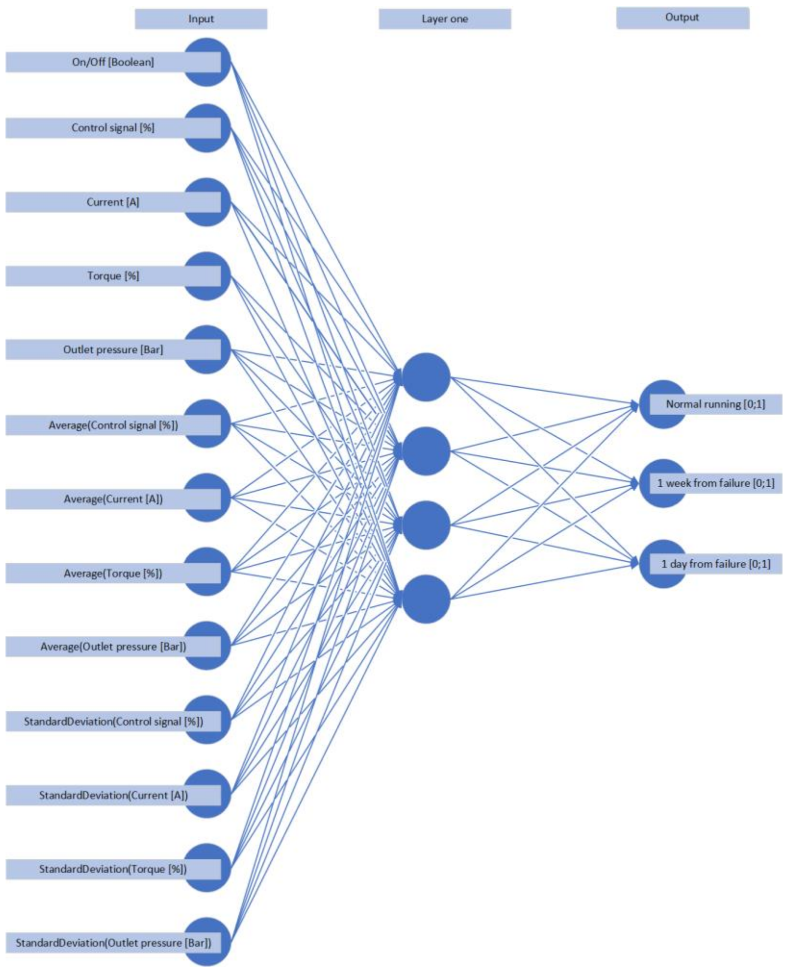

Input
Layer one
Output
On/Off [Boolean]
Control signal [%]
Current [A]
Torque [%]
Outlet pressure [Bar]
Average(Control signal [%])
Average(Current [A])
Average(Torque [%])
Average(Outlet pressure [Bar])
StandardDeviation(Control signal [%])
StandardDeviation(Current [A])
StandardDeviation(Torque [%])
StandardDeviation(Outlet pressure [Bar])
Normal running [0;1]
1 week from failure [0;1]
1 day from failure [0;1]

Figure 5.5 Model structure for LSTM model 8

The resulting confusion matrix can be seen in Figure 5.6 where close to only *Normal* was predicted by the model. The exception being 0.8% of the samples which was labelled *one day from fail* were predicted as *one week from fail*. While the number of neurons was very low, the results were not any better than previous models, here too the validation accuracy was above 99%.

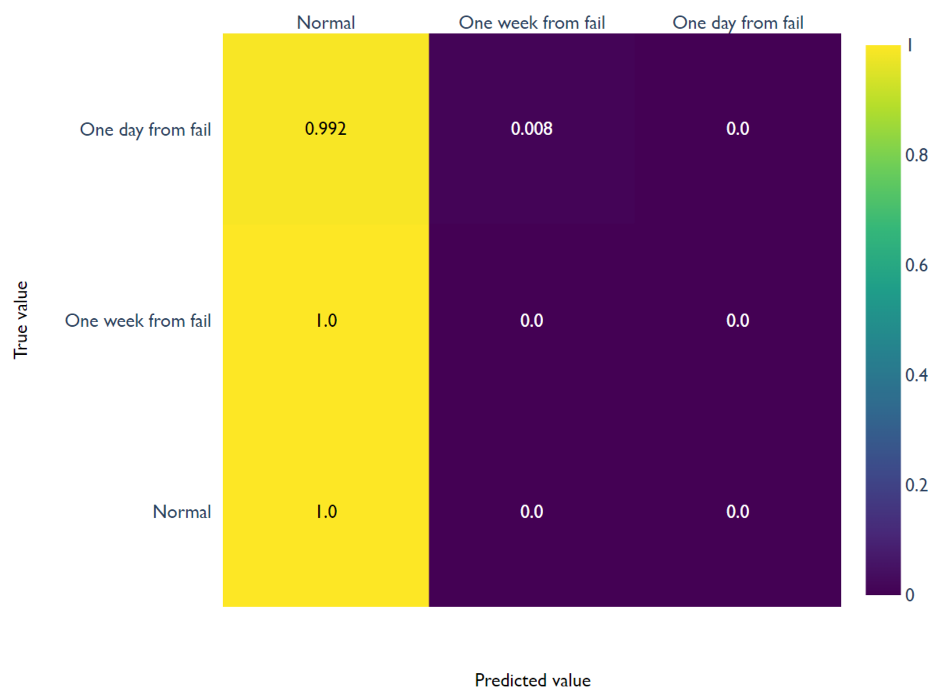


Figure 5.6 Confusion matrix for LSTM model 8

As going down on the number of neurons didn’t seem to help, going up to how the previous project modelled the system may be better as good results was achieved with this model.

Model number 10 is the closest to what was made in the previous project, although the result seems rather different. The model has the raw input features, On/Off [Boolean], Control Signal [%], Current[A], Torque [%] and Outlet Pressure [Bar]. It has two LSTM layers, the first with 32 neurons, and the second with 16 neurons as illustrated in Figure 5.7 and it has a dense output layer. The sequence length is 120 which corresponds to one hour which is the process system cycle time.

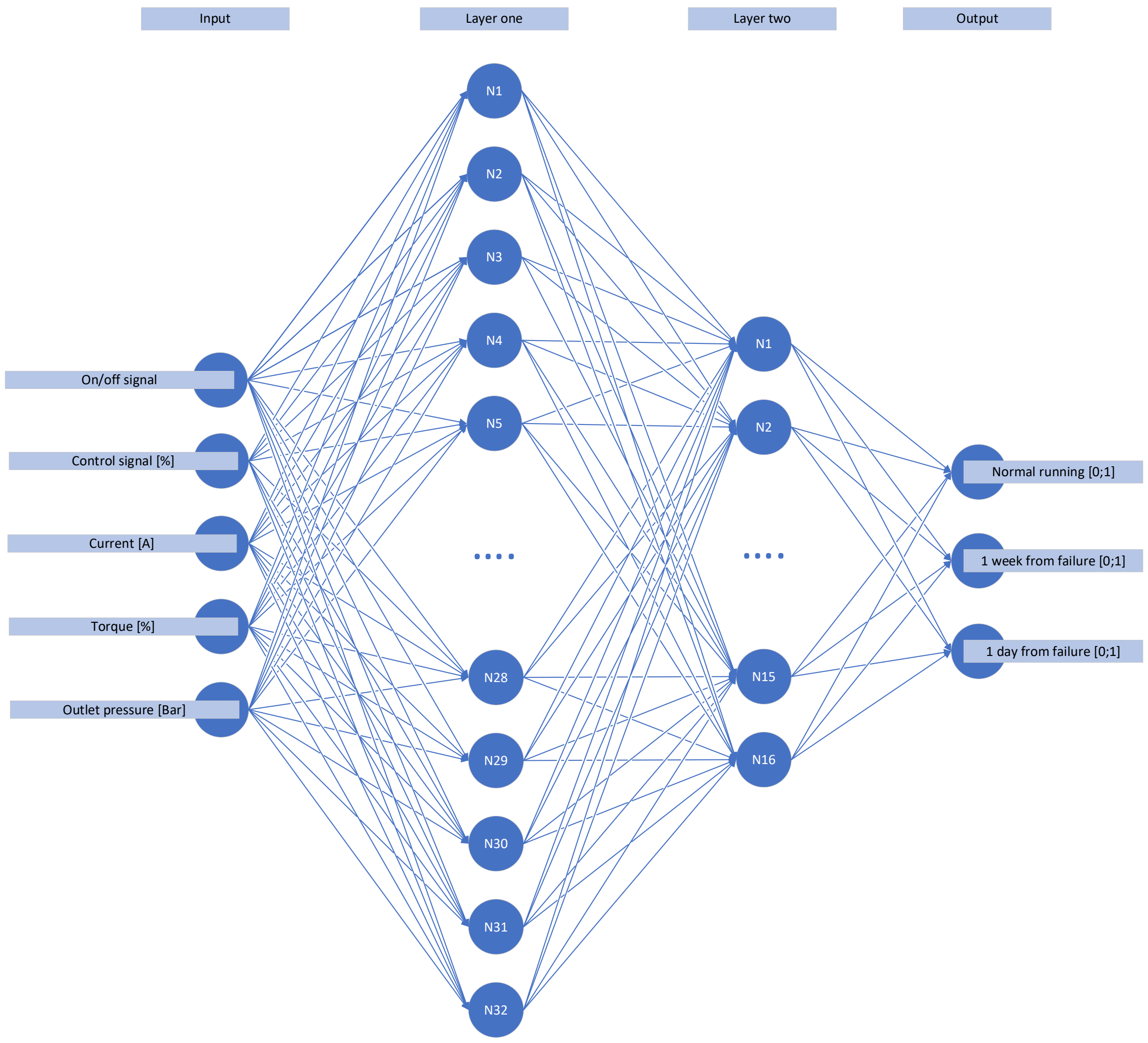


Figure 5.7 LSTM model 10, 2 layers, 32 neurons in first layer, 16 in second layer

Figure 5.8 shows the results from model 10 where 100% of the samples considered *normal* has been correctly predicted to be *normal*. 0.1% of the *one week from fail* has been correctly predicted while the remaining 99.9% was predicted as *Normal*. The label *one day from fail* was never predicted, and all of these samples ended up incorrectly as *normal*.

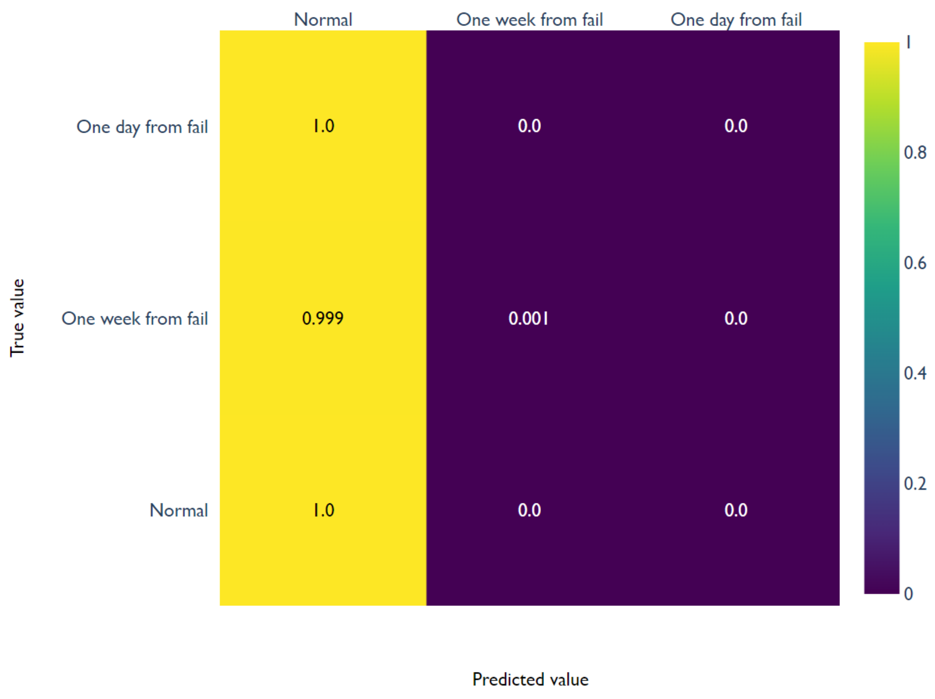


Figure 5.8 Confusion matrix for LSTM model 10

Model 10 was set up as in the previous project. The previous project however tries to predict the state of one pump, this project tries to do the same for eight pumps. There may be complexities that did not occur with one pump but occurs when there are several pumps to model.

Model 11 uses raw, average and standard deviation of the data in a three-layered model having 32 neurons in the first and second layer and 16 in the last LSTM layer. The idea being that with more neurons, the model will be able to learn many types of faults and identify them as *normal*, *one week from fail*ure or *one day from failure*. The model structure can be seen in Figure 5.9.

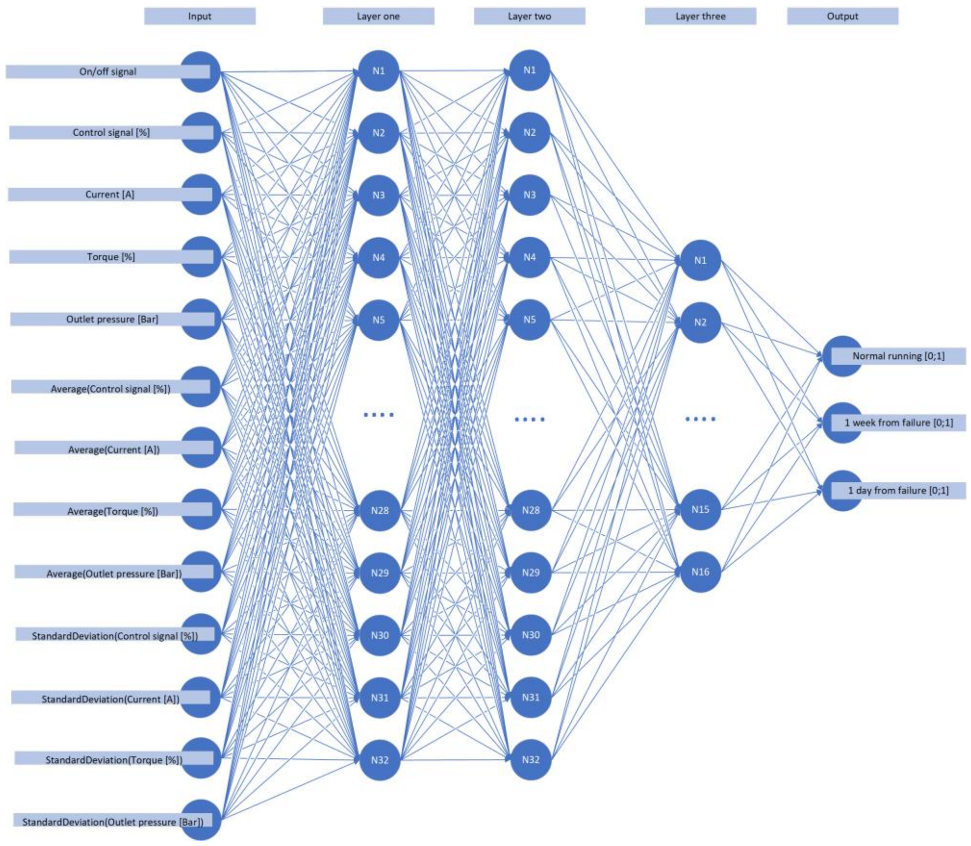


Figure 5.9 Structure for model 11

The confusion matrix in Figure 5.10 shows that creating a large network does not seem to give any better predictions where most predictions again has landed as *Normal*, with 3.2% of the *one week from fail* being correctly predicted and 0.1% of *one week from fail* has been predicted as *one day from fail*.

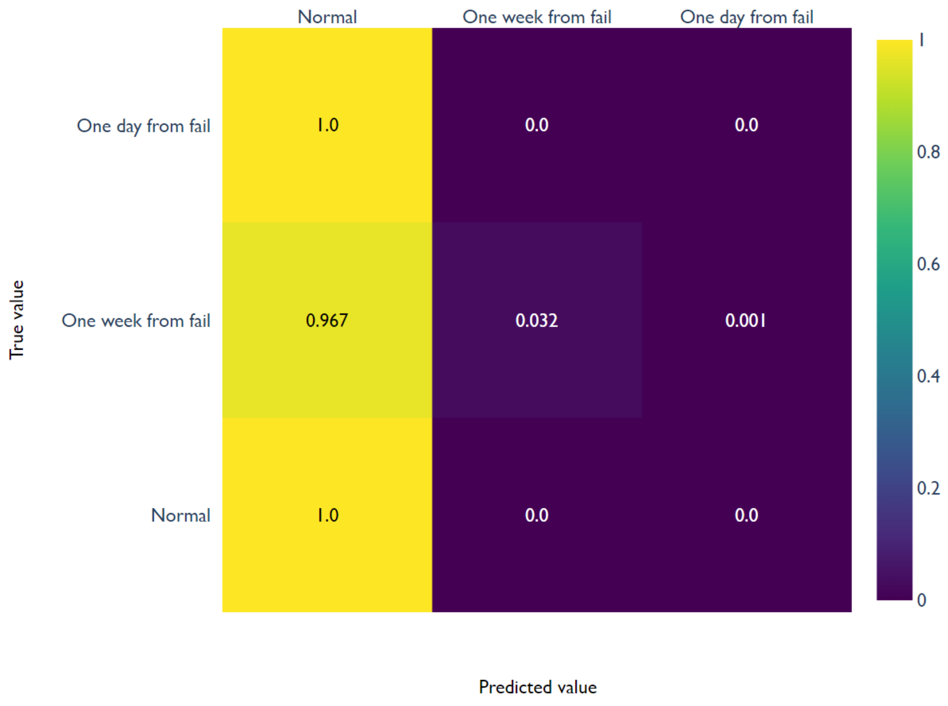


Figure 5.10 Confusion matrix for LSTM model 11

The remaining experiments are subtle variations of what has been seen, and these generally has poorer performance. All LSTM experiments and the results of them can be seen in Table 5.1, the confusion matrices for all the LSTM models can be seen in Appendix C.

Table 5.1 LSTM model results

Model number	Accuracy: Normal	Accuracy: One week from fail	Accuracy: One day from fail
1	100%	17.6%	0.2%
2	100%	0.2%	0%
3	100%	0%	0%
4	100%	0.2%	0%
5	100%	14.7%	0%
6	100%	1.4%	0.5%
7	99.8%	1.5%	15.6%
8	100%	0%	0%
9	100%	0.1%	0%
10	100%	0.1%	0%
11	100%	3.2%	0%
12	100%	0%	7.8%
13	100%	0%	0%
14	100%	0%	9.4%
15	100%	2.9%	0%

5.2 Gated recurrent unit

The gated recurrent unit has the exact same data pipeline as LSTM as shown in Figure 4.30 except for the content of train LSTM model which is now a GRU model instead of LSTM model. Table 4.10 showed the GRU experiments, some of the experiments will be investigated in more detail below.

Figure 5.11 shows the inputs, number of layers, number of neurons and the outputs for GRU model two. This is not an exact representation of a GRU model, here the inside of a layer is a representation of the number of neurons inside each of the neural networks in the GRU model. The layers represent layers of GRU cells as described in section 1.6.2. Each GRU cell in addition has a recurrence where each GRU cell iterates as many times as the sequence length. For GRU model 2, this is 128 times.

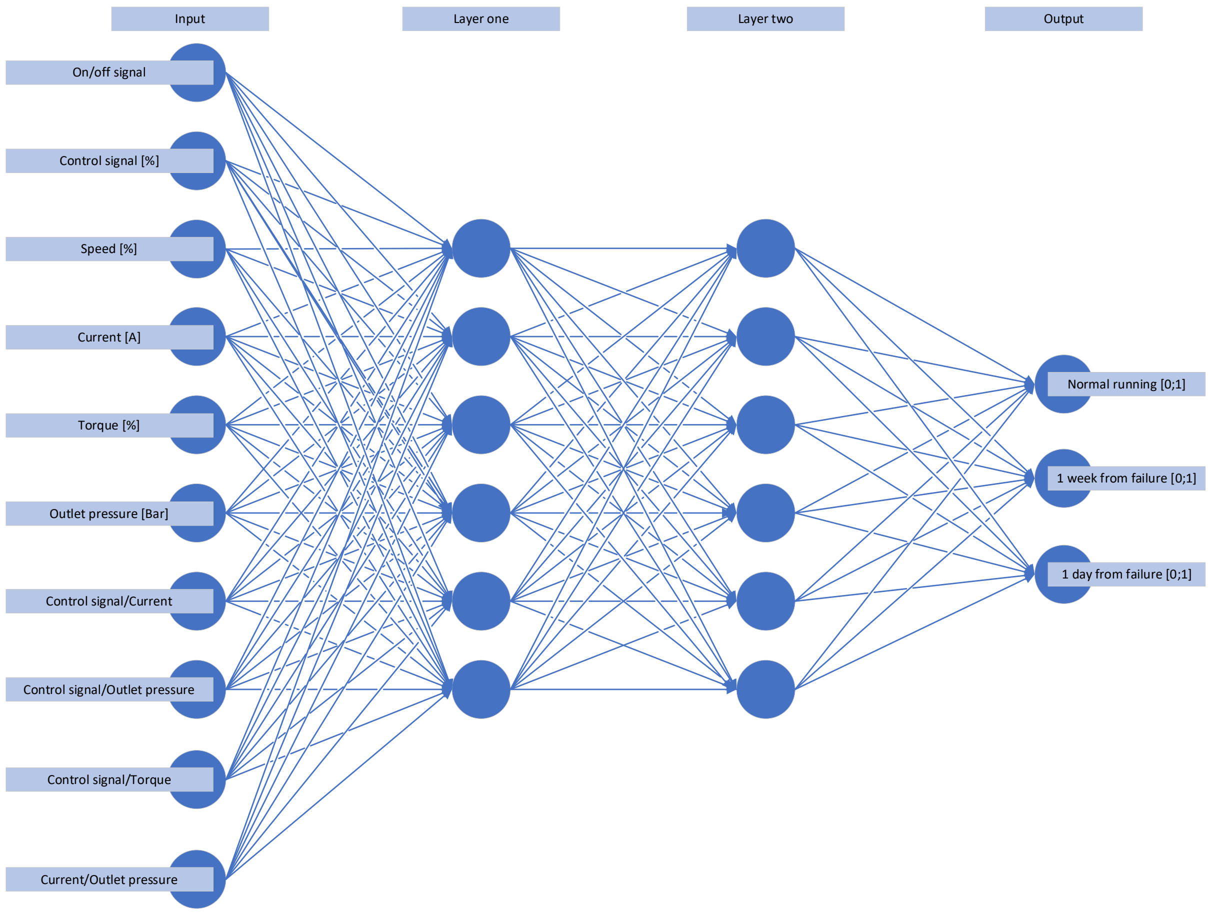


Figure 5.11 Representation of GRU model 2 as a neural network. Figure 1.6 shows that there are three neural networks for each GRU block, meaning that there are three parallel layers in the block (with varying weights).

Figure 5.12 shows the result for GRU model 2 in a confusion matrix. The confusion matrix has two axis, one which is the true value from the test set, and one which is the predicted value from the model. Each row of the confusion matrix is a distribution which adds up to 1 where 1 indicates 100% of the predictions. I.e., in Figure 5.12 100% of the *Normal* operations were correctly predicted. Only 1.2% were correctly predicted in the *One week from fail* category, while 0.9% of the true *one week from fail* was predicted as *One day from fail* and the remaining 97.9% of the *One week from fail* were predicted as *Normal*. Similarly on the top row for the true *One day from fail*, 0% were correctly predicted, 0.6% was predicted as *one week from fail* and 99.4% was predicted as *Normal*.

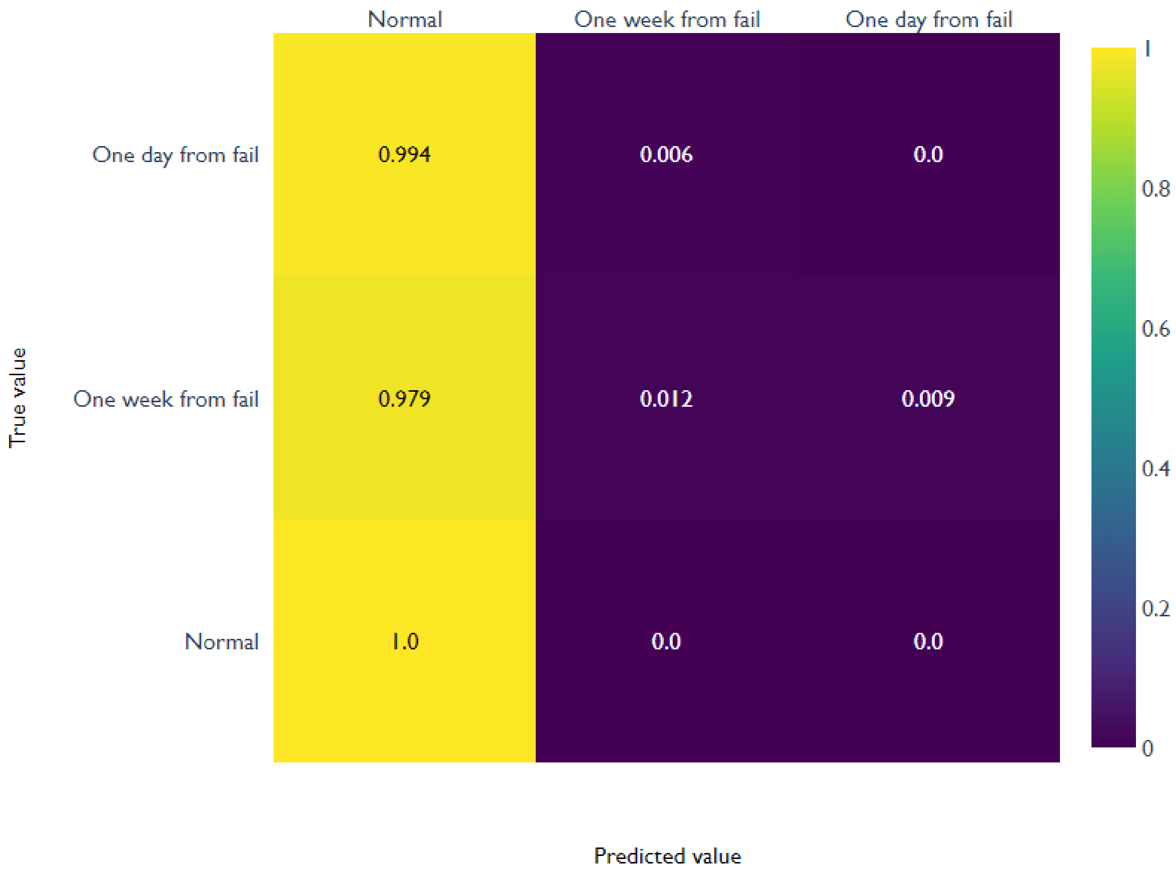


Figure 5.12 Confusion matrix for GRU model 2

The GRU model 4 uses the average and standard deviation in addition to the raw and calculated values as seen in Figure 5.13. This model also has two layers, however there are also more input features and more neurons in the first layer.

The model gets 100% right for the *normal* operations, 13.2% right for *one week from failure*, falsely predicting 4.1% of the *one week from failure* as *one day from failure*. The remaining 82.8% are predicted as *normal*. 3.3% of the *one day from failure* are predicted correctly, 5.6% are wrongly predicted as *one week from fail* and the remaining 91.1% are predicted as *normal* in Figure 5.14.

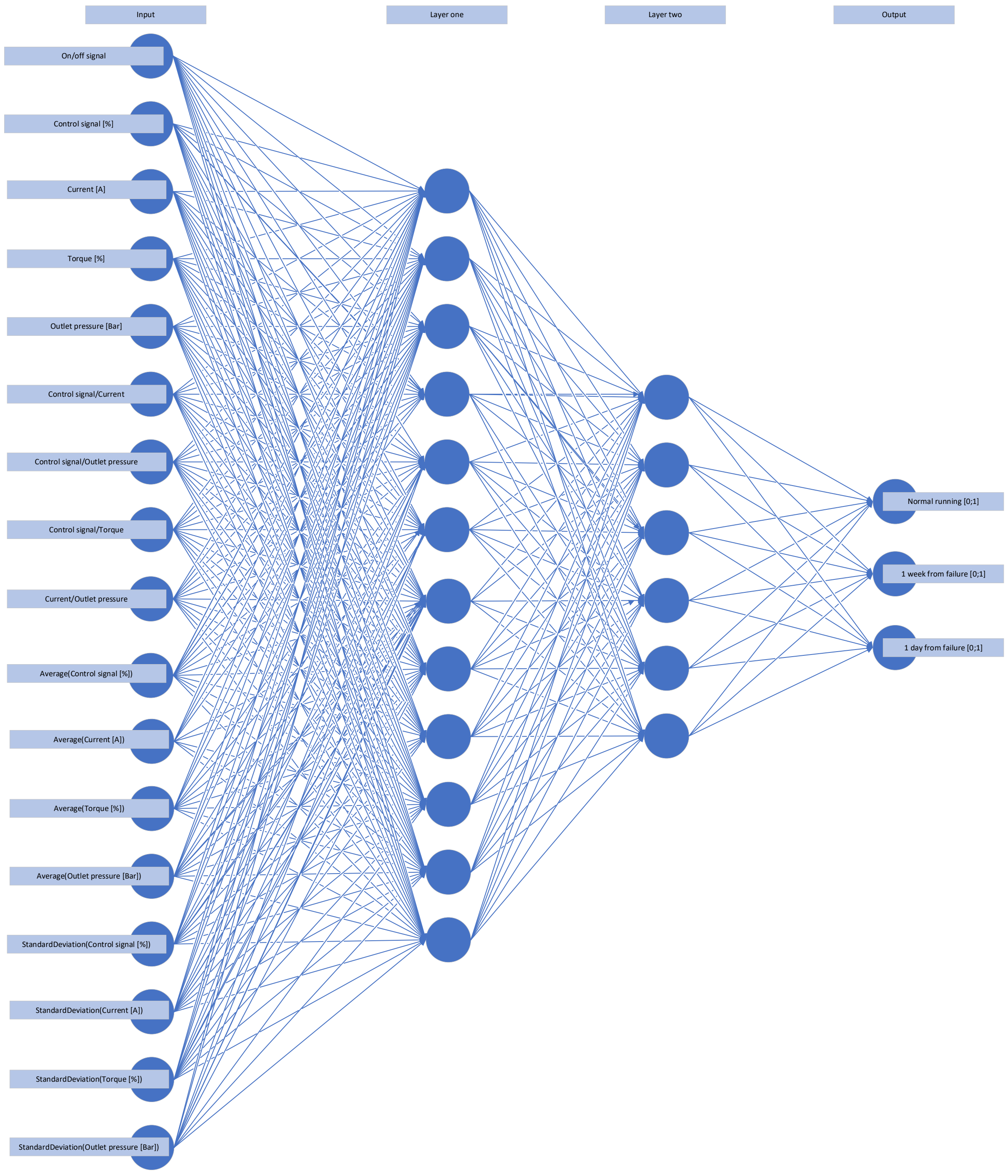


Figure 5.13 Representation of GRU model 2 as a neural network.

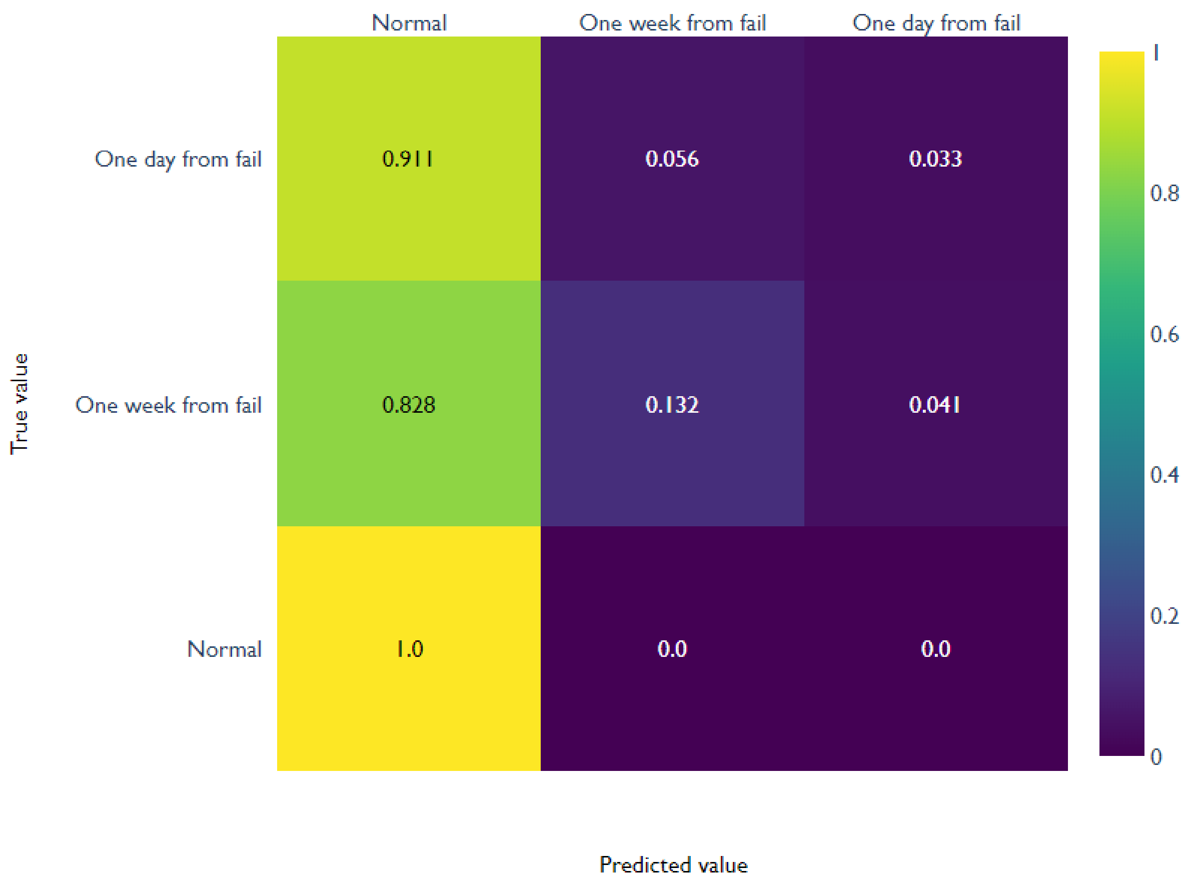


Figure 5.14 Confusion matrix for GRU model 4

The gated recurrent unit model 6 structure can be seen in Figure 5.15. This model only uses aggregated data. There are two aggregations, average and standard deviation. Both aggregations have an aggregation time of 22 minutes which approximately adds up to six hours for each sequence. The model has only one layer with 6 neurons. Additional this model is stateful, meaning its cell state is transferred between sequences.

This model also correctly predicts *normal* operations 100% of the tests. 33.7% of the *one week from fail* is predicted correctly and the remaining 66.3% are predicted as *normal*. One *day from failure* is never correctly predicted, 3.1% of them are predicted as *one week from failure* and 96.9% are predicted as *normal*.

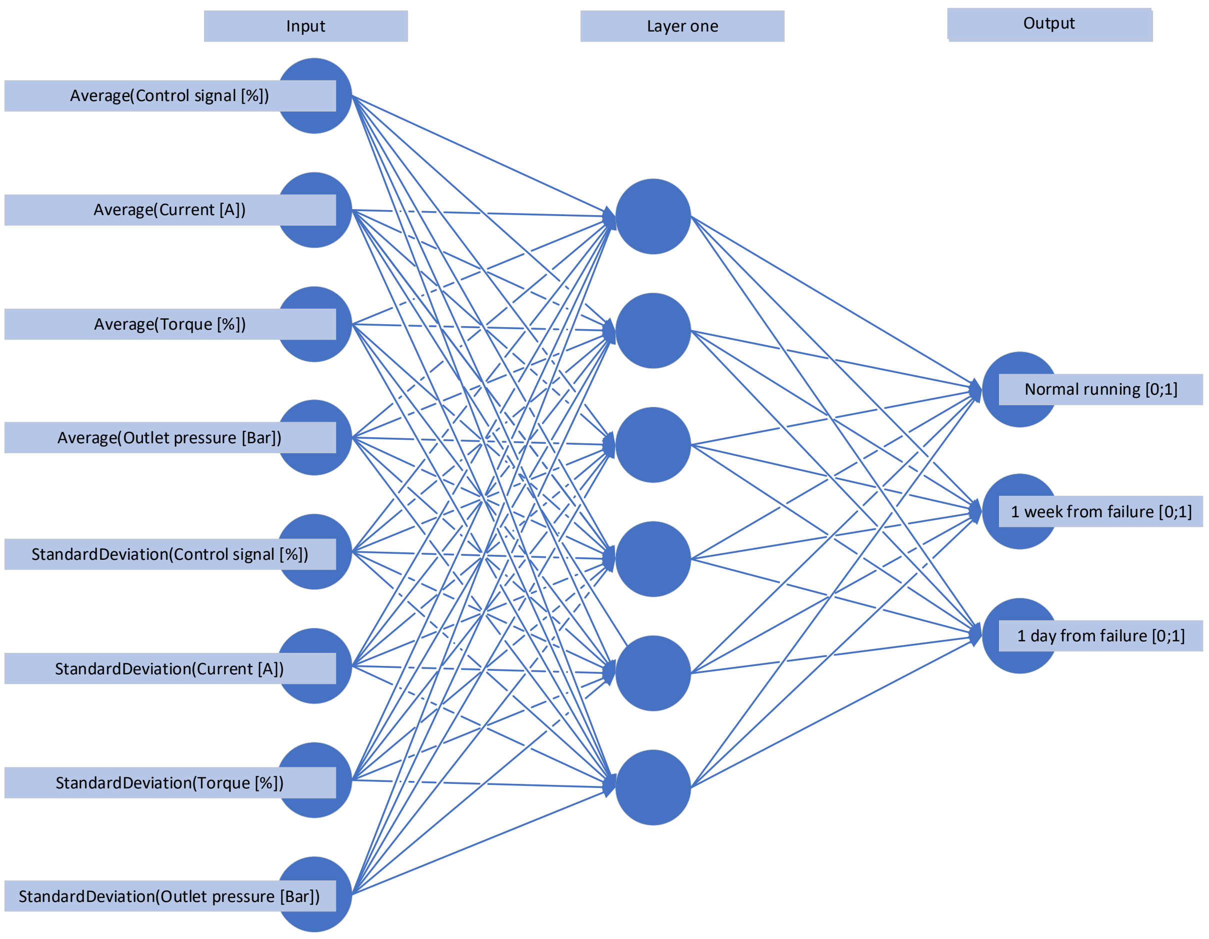

Input
Layer one
Output
Average(Control signal [%])
Average(Current [A])
Average(Torque [%])
Average(Outlet pressure [Bar])
StandardDeviation(Control signal [%])
StandardDeviation(Current [A])
StandardDeviation(Torque [%])
StandardDeviation(Outlet pressure [Bar])
Normal running [0;1]
1 week from failure [0;1]
1 day from failure [0;1]

Figure 5.15 Representation of GRU model 6 and 7 as a neural network.

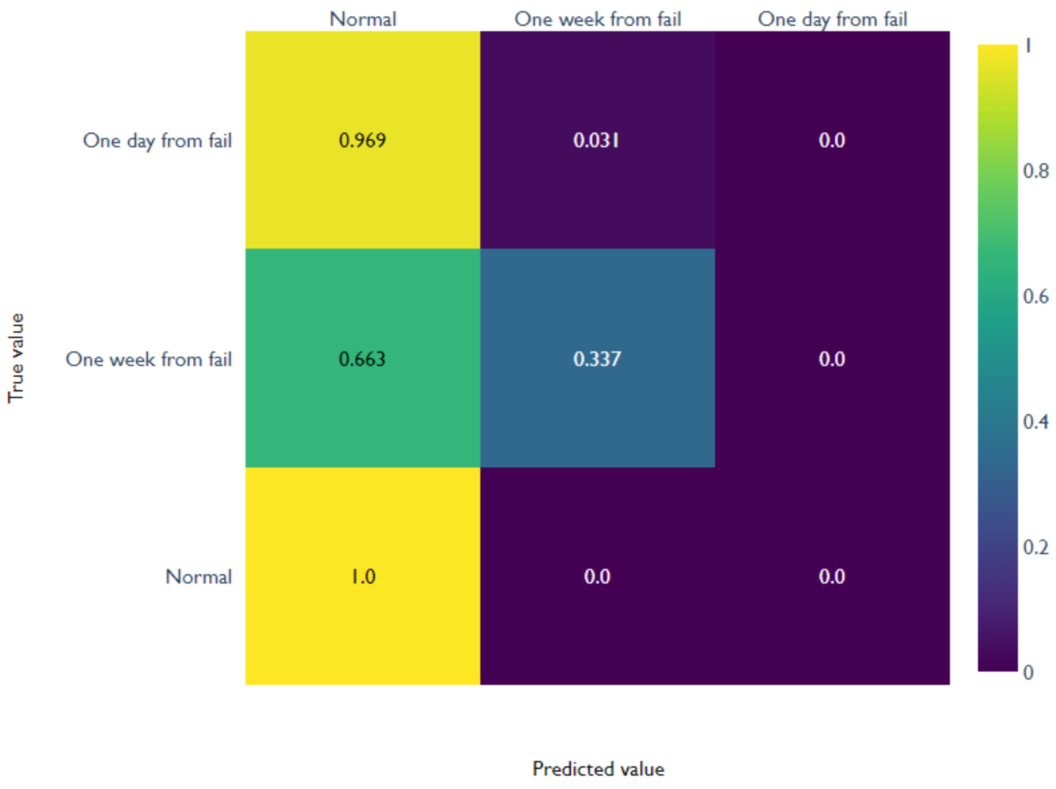


Figure 5.16 Confusion matrix for GRU model 6

GRU model 7 has the same structure as GRU model 6 although the aggregation is done over one hour and keeping the sequence equally long. This means each sequence represents 16 hours of data. The results seem to be improving in Figure 5.17 where 100% of the *Normal* labels are correctly being predicted. 50% of the *one week from fail* is correctly predicted and the remaining 50% are predicted as *Normal*. *One day from fail* is correctly predicted 31.2% of the samples and the 68.8% remaining are predicted as *Normal*.

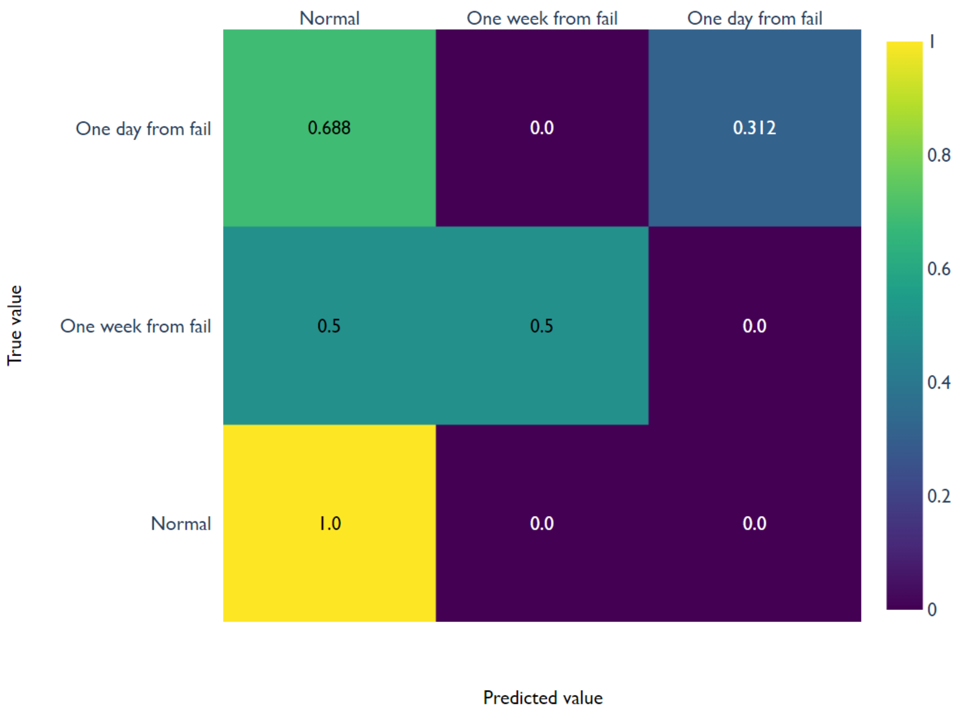


Figure 5.17 Confusion matrix for GRU model 7

The remaining experiments are subtle variations of what has been seen, and these generally has poorer performance. All GRU experiments and the results of them can be seen in Table 5.2, the full results in confusion matrices can be seen in Appendix D.

Table 5.2 GRU experiments

Model number	Accuracy: Normal	Accuracy: One week from fail	Accuracy: One day from fail
1	100%	0.7%	0%
2	100%	1.2%	0%
3	100%	0.3%	0.8%
4	100%	13.2%	3.3%
5	100%	0.3%	0.0%
6	100%	33.7%	0%

7	100%	50%	31.2%
8	100%	0%	0%
9	100%	0%	0%
10	100%	0.4%	0%
11	100%	1.8%	0%
12	100%	0.8%	1.3%
13	100%	0.6%	0%
14	100%	1.2%	0.4%
15	100%	3.1%	0%
16	100%	0%	6.2%

5.3 Support vector machine results

The SVM experiments were introduced in Table 4.11. Some of the experiments will be investigated further in this chapter. As was explained earlier, SVM does not allow partial fitting, thus what pumps were trained on was not stored in the same manner as in GRU and LSTM. SVM is tested on the pumps that GRU model 3 has not been trained on, meaning it will use the same test set as GRU model 3.

SVM model one uses the raw features On/Off [Boolean], Control Signal [%], Current [A], Torque[%] and Outlet Pressure [Bar], the aggregated features of the raw features, except for On/Off for both average and standard deviation. It also uses the calculated features based on the raw features seen in Figure 5.18. The model uses the Radial Basis function as the kernel function with parameters C as one and gamma as one.

Control signal/Current

Control signal/Outlet pressure

Control signal/Torque

Current/Outlet pressure

Figure 5.18 Calculated features

Figure 5.19 shows the resulting confusion matrix. 99.3% were correctly classified as *Normal*, while 0.7% of the true *Normal* was classified as *one week from failure*. 4.5% of the *one week from failure* was correctly predicted while 95.5% of them was misclassified as *Normal*. *One day from fail* was never predicted, 4.3% of them was predicted as *one week from failure* and the remaining as *Normal*.

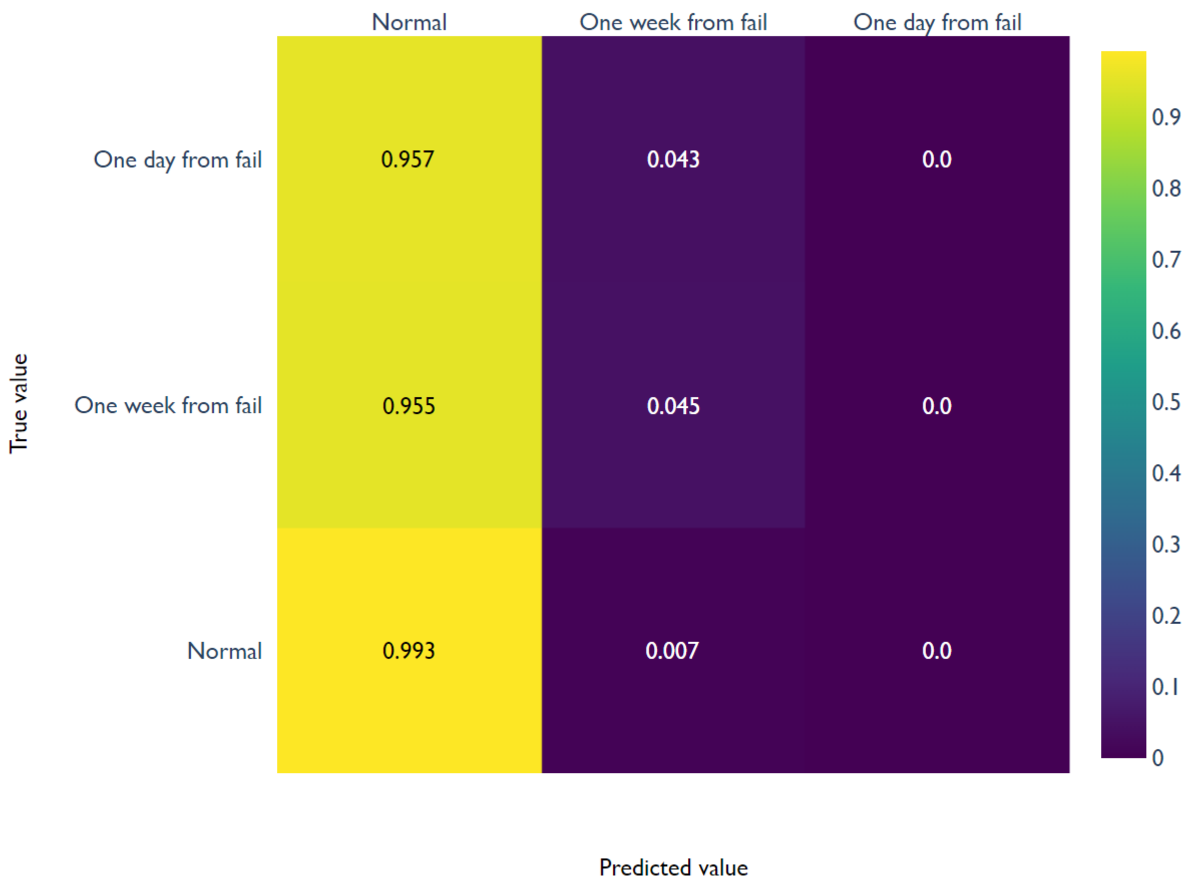


Figure 5.19 Confusion matrix from SVM model 1 using radial basis function

A grid search using cross validation was tested in SVM model 2, the range seemed to be off as validation accuracy was very poor with a total accuracy of 4.9% and all predictions became *one day from fail*. Using grid search also proved to be very time consuming, using several days for one result. Therefore, several experiments were tested near the parameters SVM model 1. This was done sequentially by tuning one parameter at the time. Figure 5.20 shows the C parameter along the x-axis, gamma as bars on the y-axis and the validation accuracy of the three labels as a line on the y-axis. The validation accuracies appear to be rather good, with mostly above 90% for *normal*, above 60% for *one week from fail* and around 30% for *one day from fail*. However, this was only on validation set, that is, validating with the same pumps that it was trained on. When testing on new pumps however, only one label is predicted per SVM model in SVM models 16, 17, 18 and 19. The models 16, 17, 18 and 19 used the raw, average and standard deviation features.

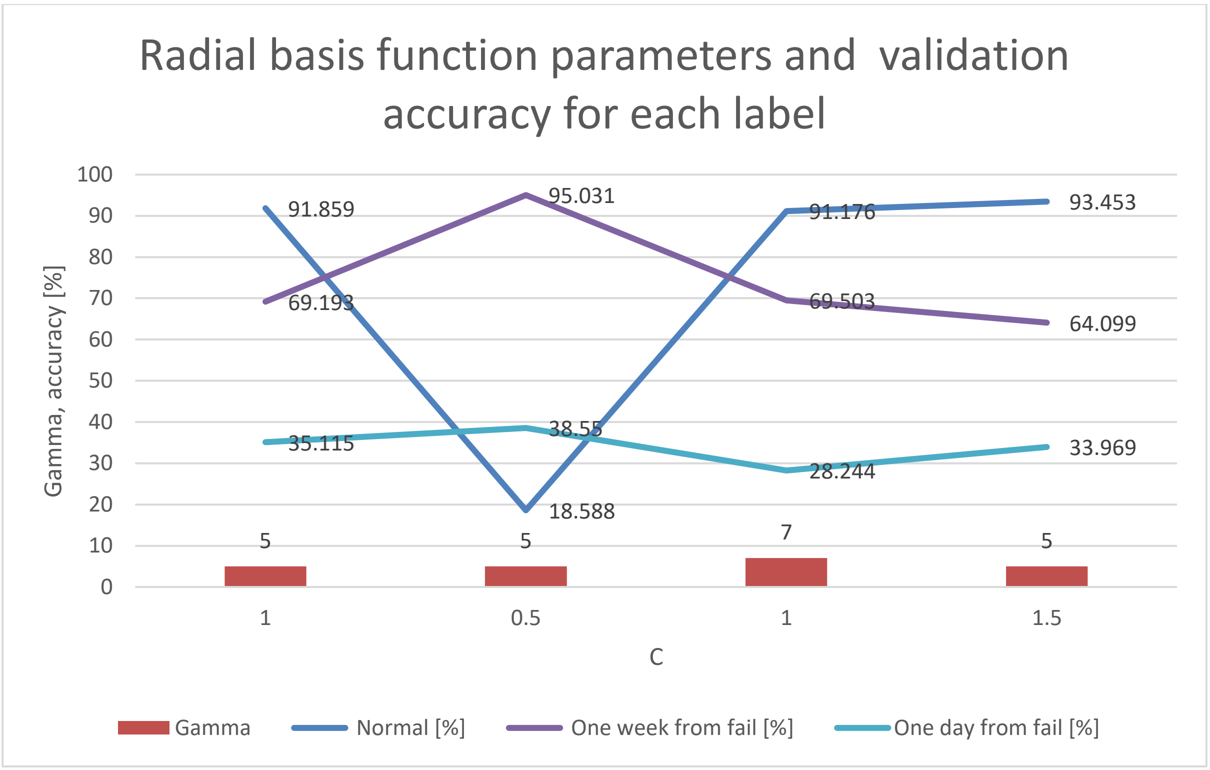


Figure 5.20 Validation accuracy for four support vector classifiers using radial basis function as kernel

Using the standard parameters for SVM's radial basis function C as one and gamma as 'scale' the results became somewhat different, although not necessarily better. SVM model 20 used the raw features, average and standard deviation and the calculated features. The results are shown in Figure 5.21 where 57.6% of the *normal* samples were correctly classified, 0.7% of them was classified as *one week from fail* and 41.7% was classified as *one day from fail*. *One week from fail* was correctly predicted 3.8% of the samples, 28.7% was predicted as *normal* and 67.5% was predicted as *one day from fail*. *One day from fail* had a rather high correct prediction rate of 68.9% where 5.5% of the remaining were predicted as *one week from failure* and the last 25.6% was predicted as *Normal*.

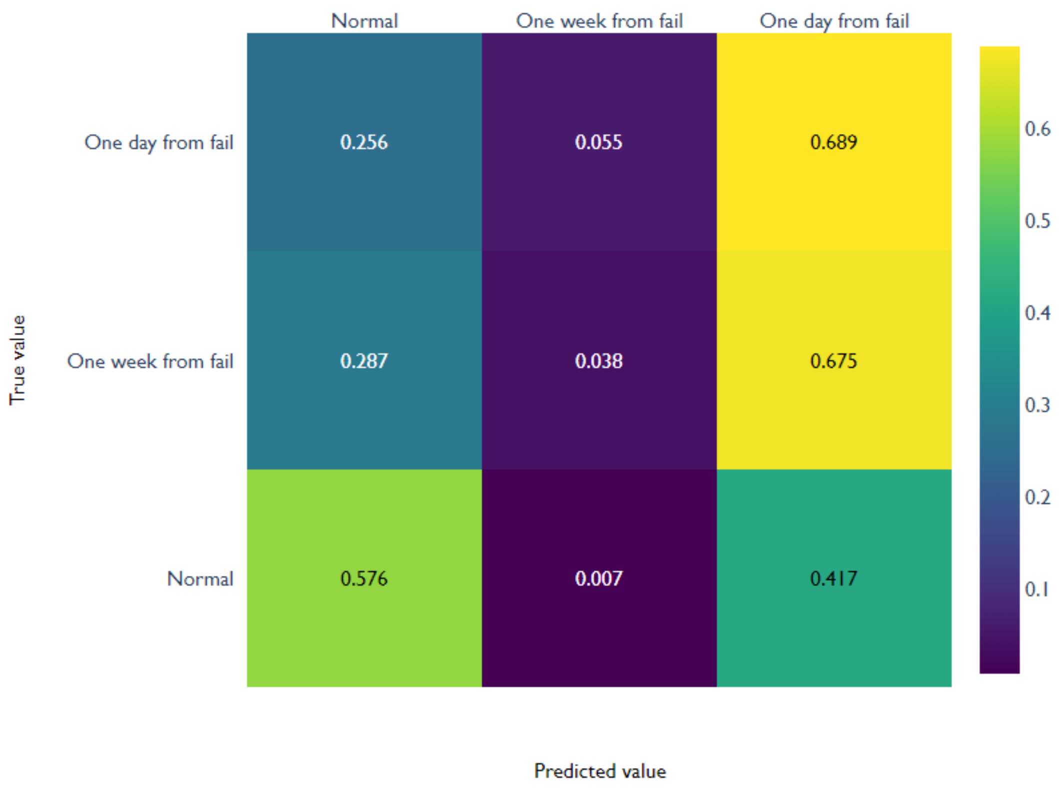


Figure 5.21 Confusion matrix for model 20

The linear SVM model 3 used raw, average, standard deviation and calculated features and set C to default as one. The aggregation time for average and standard deviation was 22minutes. The confusion matrix in Figure 5.22 shows that most samples were put into *Normal* with a few samples landing in *One day from failure*, and *One week from fail* was never predicted.

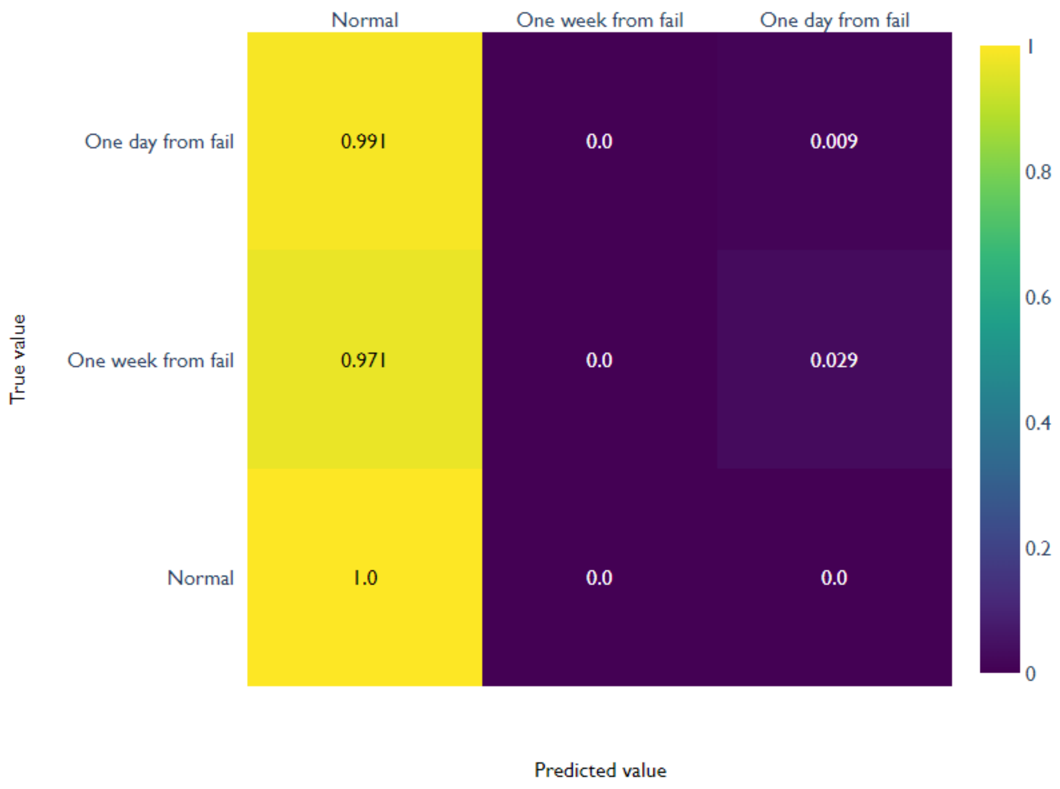


Figure 5.22 Confusion matrix for SVM model 3

SVM model 6 uses a polynomial kernel function of grade 3, with C as 1 and gamma as 1. It also uses the raw features, average standard deviation and calculated features. The confusion matrix shown in Figure 5.23. Once again, most samples were put into one class, here the one class became *One day from fail*.

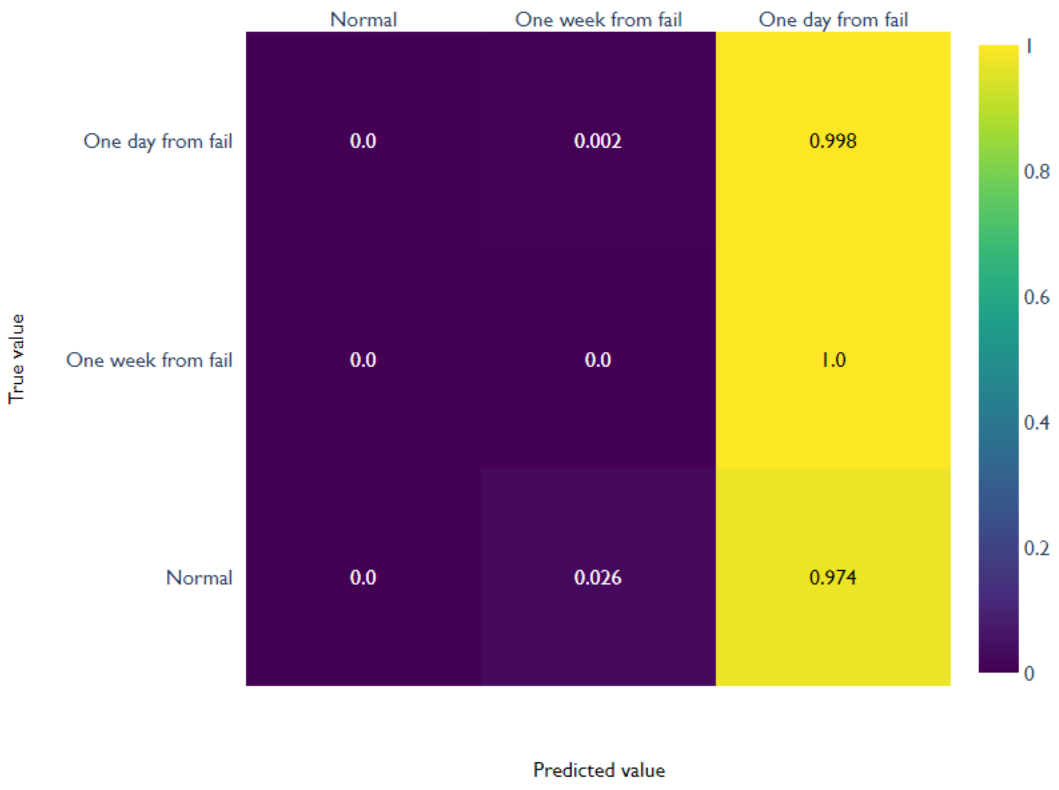


Figure 5.23 Confusion matrix for SVM model 6

SVM model 7 uses the sigmoid kernel with standard parameters, C as one, gamma as one and coef0 as 0. The model uses raw, average and standard deviation as features. It does for the most part also predict one class as seen in Figure 5.24. The figure indicates that between 80-90% of the samples are predicted as *one day from fail* no matter the true label. 10-14% are predicted as one week from *normal* and 0-3% are classified as *normal* independent of the true label.

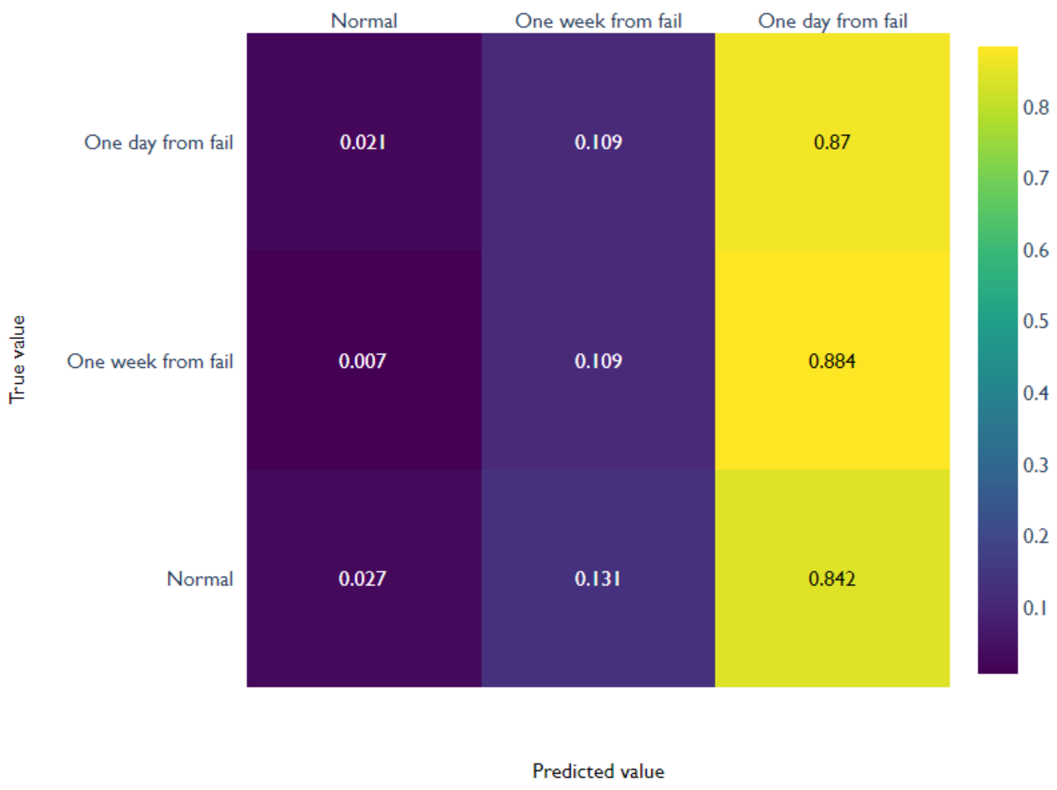


Figure 5.24 Confusion matrix from SVM model 7 using sigmoid kernel function

The remaining results can be seen in Table 5.3, the confusion matrix of each successful model can be seen in Appendix E.

Table 5.3 SVM experiments

Model number	Accuracy: Normal	Accuracy: One week from fail	Accuracy: One day from fail
1	99.30%	4.50%	0%
2	0%	0%	100%
3	99.9%	0%	0%
4	Failed	Failed	Failed
5	Takes too much time	Takes too much time	Takes too much time
6	0%	0%	99.8%
7	2.7%	10.9%	87%
8	Takes too much time	Takes too much time	Takes too much time
10	0%	0%	100%
11	0%	0%	100%

12	0%	0%	100%
13	0%	0%	100%
14	0%	0%	100%
15	2.9%	0%	96.8%
16	100%	0%	0%
17	0%	100%	0%
18	100%	0%	0%
19	100%	0%	0%
20	57.5%	37.8%	68.9%

6 Discussion and future work

This chapter will discuss various choices made during the thesis, what has been learned, issues that has occurred and future work based on the discussion.

6.1 Input data

There is a concern that not all failures have been found. Figure 6.1 shows a large difference in how the pump behaves. This is very much seen in the control signal and the current with large variations before there is a gap in samples. After the gap, the variation on both is drastically reduced. This all happens while the outlet pressure is somewhat stable. In section 4.3.2 it has been seen that the current increases in variation when the control signal increases, however a cause for the high variation in control signal is not found. It may be caused by the pump having been degraded and replaced, or the pump before being degraded or some entirely different reason. This adds to the point of having good documentation of what happens to the pumps, or more generally to the factory.

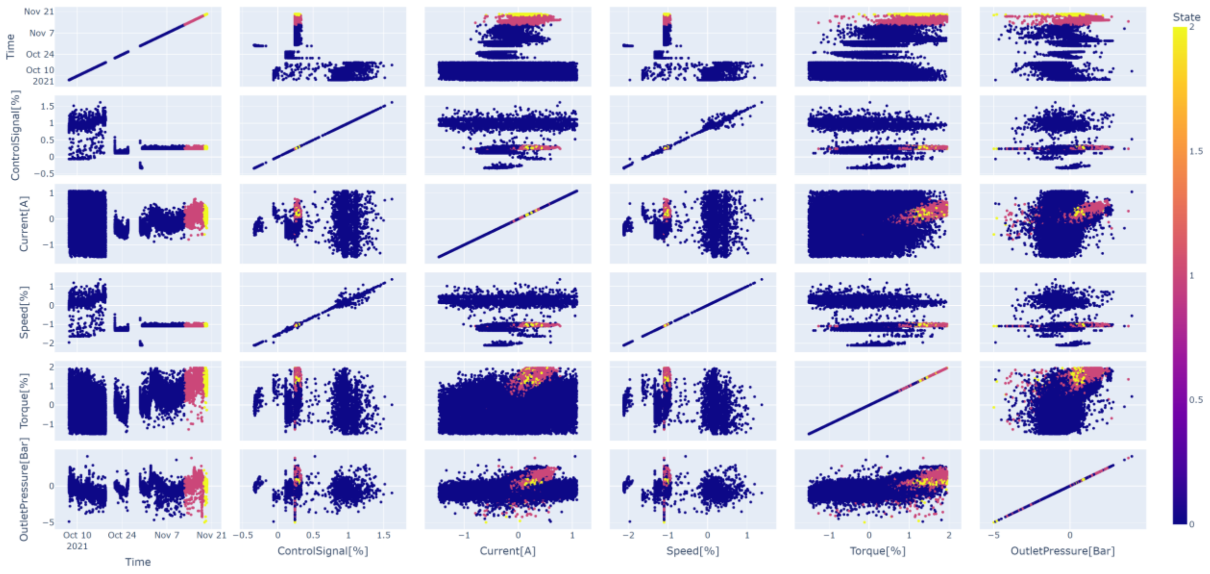


Figure 6.1 Pump 19 07.10.2021 to 19.11.2021

There is a data set available which measures the same variables used in this thesis every 50ms for all pumps. Due to processing times and different storage format, this was omitted. To use this dataset, part of the code for retrieving data must be updated. Much of the underlying code exists.

6.2 Design of output classes

Many of the models predict mostly one class, this may be caused by the model structure, the input data, but also how the classes are defined in regard to the input data.

The labels are set somewhat arbitrarily when considering that there are many faults. The labels are purely based on time and the assumption that the last week of a pump's lifetime should look different from earlier. It may be that this is true to some degree, however degradation usually happens over time, causing the measurements to change over time. Depending on what time a fault, which is not purely a degradation, start to occur, the measurements related to a fault may look different. In addition, depending on the fault the measurements may look different. It is possible that the model learns to recognize many types of faults as *one week from failure* or *one day from failure* given a complex enough model. However, it may give more information and require less complex models to create models dedicated to each type of fault which can predict the labels *normal*, *one week from failure* and *one day from failure* for each of them. It may be that some faults are not captured by the current measurements, i.e., a partly broken seal may cause the operators to temporarily shut down a pump and use a backup pump, which for the current labelling would be the same as any other fault. However, it may be that the differences in the measurements are not noticeable.

The data to model each individual fault is not yet available but is something to strive towards in general for future similar projects.

More manual work could be done in classifying similar faults with the goal to create smaller models which monitor individual faults. When a failure occurs, the type of fault would be identified.

In future works, it may be wise to investigate if there are particular faults that is being detected by the models.

6.3 Data versioning

Although the time-series have few changes, the labelling data has changed during the project. Some have been removed, some have been updated due to new information. This should have been logged to keep control on what models, if any, has been trained on other than the latest version of the labels.

6.4 Features

6.4.1 Combined features

It has been discovered that the calculated values in Figure 6.2 did not work as intended as these were divided after scaling causing them to often become very high. When features have high magnitude values compared to other features, the high magnitude features are likely to dominate the outcome of the model. Models using the calculated values may have less impact from the other features, most experiments were set up using them. When discovered how they behaved they were omitted from further models. An example of the scale achieved with the calculations can be seen in Figure 6.3 where the calculated features range from -1500 to 2500 and the raw features typically range from -5 to 5.

In future projects, the calculated features may be feasible, however the fraction will have to be made before scaling.

Control signal/Current
Control signal/Outlet pressure
Control signal/Torque
Current/Outlet pressure

Figure 6.2 Calculated values

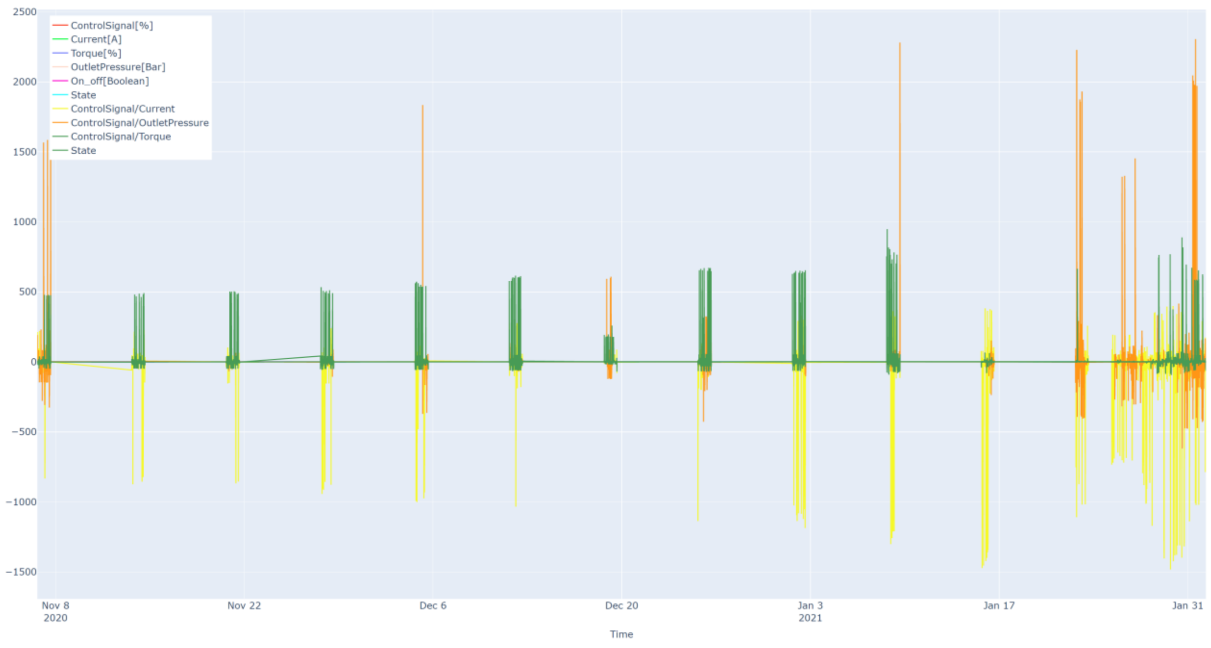


Figure 6.3 Line plot of calculated values

6.4.2 Combination of aggregated and non-aggregated data

An idea of combining two sequences of different time scale was introduced in section 4.4.2. The idea was to use one hour of data to represent a detailed view of the pump, and to also use the previous 24 hours as distinct features in the LSTM and GRU models. The LSTM and GRU models both use sequences and the idea was to use one hour of non-aggregated data together with aggregated data which would cover the previous 24 hours. The aggregated data was instead wrongfully joined with the non-aggregated data based on timestamps, meaning that the closest aggregated datapoint was joined to many non-aggregated data points as seen in Figure 6.4. When attempted to solve this proved to be rather tedious as each sequence would have to query the database for the correct data and several changes would be required in the code structure. As the aggregated data has proved to sometimes do well, this may still be of interest to pursue.

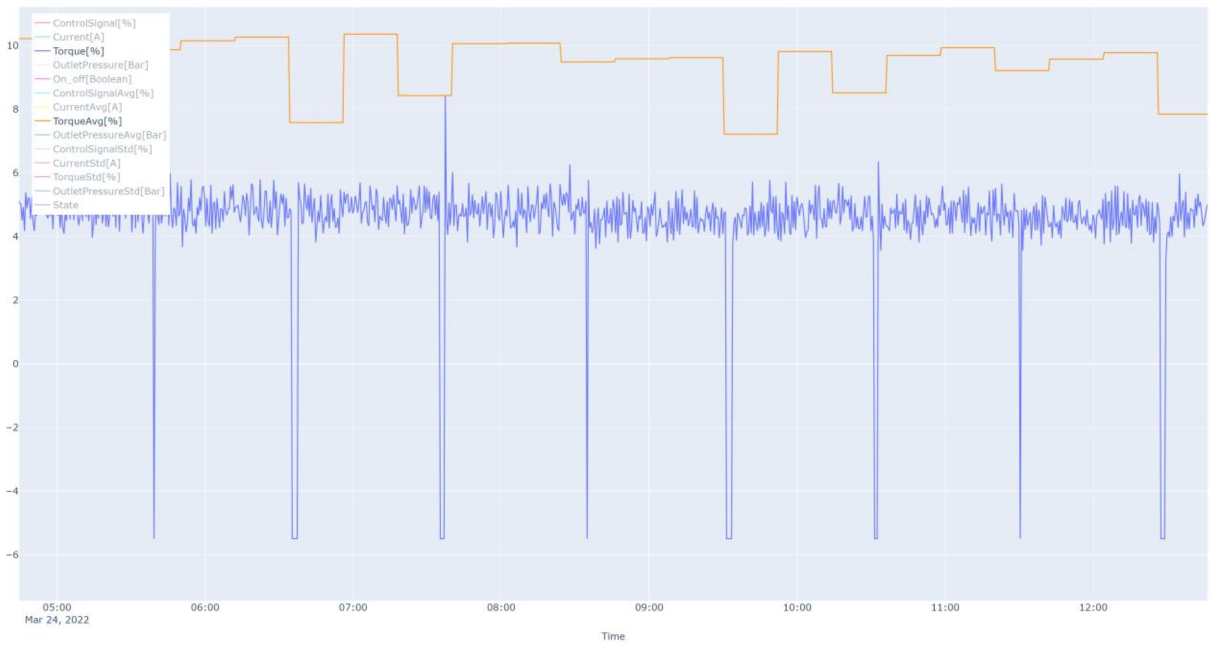


Figure 6.4 Raw data and averaged data joined on timestamps

6.4.3 Outliers

The removal of outliers in section 4.4.5 didn’t seem ideal as both the current and torque had a very long tail on the negative side. It was attempted to increase the percentage of the lower quantile to remove more of the low values. This did not prove very effective, as few of the actual low values were removed. Rather it removed many values from the large distribution between -2 and 2. The quantiles were removed on a per pump basis, thus it is likely that there are one or a few pumps that have very low values that cause this long tail. Removing these may improve the SVM models.

6.5 Computer restrictions

6.5.1 Graphical processing unit

Although a graphical processing unit (GPU) was available, setting up the NVIDA CUDA software was not trivial as DLL files was not found. After trying to fix this for a couple of days, the idea was dropped. This may have sped up some of the training.

6.5.2 Memory trouble

The method used for sequencing data in LSTM and GRU was very heavy for the random-access memory (RAM) of the computer. Previously during sequencing, there was an overlap of samples. I.e., for a sequence of 120 samples, the first sequence would be sample 0-119, the second sequence would be 10-129 etc. As this demanded very much RAM to execute, the overlap was removed causing the first sequence to be 0-119, the second to be 120-239 etc. In hindsight, this has reduced the training data considerably and may be a big reason for the results achieved.

An evaluation of the impact of this method should be done in order to consider whether the training pipeline has to be changed, an alternative is of course to use another computer with more memory.

As there are more *Normal* samples than any of the other labels, a partial solution may be to make overlapping sequences of only the labels *One week from failure* and *One day from failure*. This would reduce the bias towards *Normal* and still hold memory low compared to making overlapping sequences for the entire data set.

6.6 Design of Models

This project has tried to model all the pumps in the process part seen in Figure 1.1 as one and the same model. It has taken the common measurements and assumed that the pumps behave similar to each other. They are primarily of the same pump type which does call for some similarity. They are also at various locations with separate process steps before and after. Some pumps are pumping from a tank, others are pumping in series from another pump which means that they have some effect on each other. Some are controlled manually, others are controlled by a PI controller. They for the most part also have different temperatures on the medium they pump which may have an impact on the behavior of the pumps and how they degrade.

The previous project described in section 1.2.1, got relatively good results on the LSTM model for one pump with a 78% accuracy. Through various network structures in LSTM, GRU and SVM, very few models gave good results when trying to model all the similar pumps as one model in this project. This may indicate that one model should be trained separately for each of the pump locations. Training one model for each pump location may be the easiest way forward, as the code is set up to train on one physical pump at the time based on a list of tag names. For the long short-term memory and gated recurrent unit models it is also possible to use the best trained models and train one additional round on specific pump locations. This would allow for the model to know of many types of faults and may be better calibrated to one specific tag location.

An alternative is also to model the system of pumps, this would allow the model to have more information about the surroundings of the pumps. Modelling as a system would likely require a much more complex model. A possible way to do this is to model two separate models which are separated by vessels (Buffer tank to E-11 in Figure 6.5 and Figure 6.6, and E-11 to Reactor). The reasoning being that each pump in series will have an impact on the next, and process part one and process part two being independent of each other when both are running. This alternative will however require more work than moving back to having one model per pump.

The recommended way forward is to test the performance of the best model from this project against a set of models trained for specific tag locations. During testing it should be investigated if there are certain faults that are being detected, what they are, and if there are others that aren't detected.

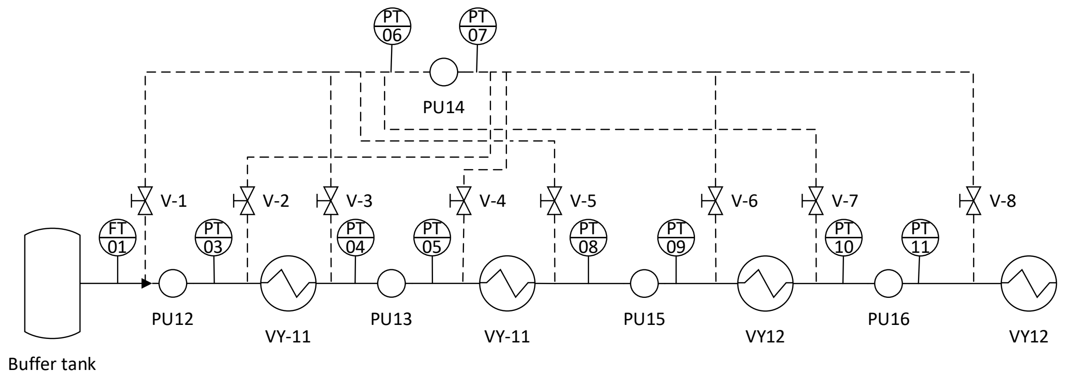
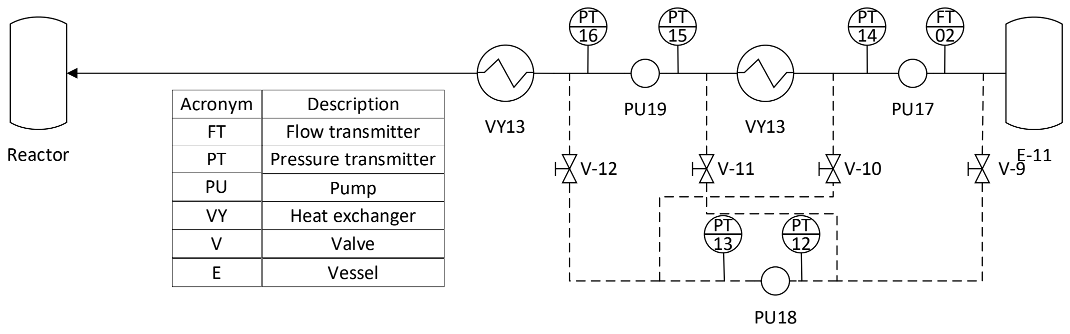


Figure 6.5 Process part one



Acronym	Description
FT	Flow transmitter
PT	Pressure transmitter
PU	Pump
VY	Heat exchanger
V	Valve
E	Vessel

Figure 6.6 Process part two

6.7 SVM

The linear, sigmoid and polynomial only got one experiment as the grid search took too much time and eventually were shut down by windows update forcing a restart. All the grid searches had the tolerance stopping criterion turned up to 0.01 from 0.001 (default) to reduce the training time. For further studies, the tolerance for the grid search may be increased more to give a coarse overview of good parameters which then can be used for fine-tuning.

6.8 Validation of training process

As few of the models have given good results, one may wonder if there is fundamentally something wrong with how the model is trained and the setup for the project. Therefore, to recapture the essence of the previous project, a model trained for only pump 19 was trained using the same training pipeline as has been used in this thesis.

The model has two layers with four neurons each and only uses the raw measurements Control Signal [%], Current [A], Torque [%], Outlet Pressure [Bar] and Control Signal [Boolean]. The sequence length is 120 which is the process cycle for the pumps. The model structure can be seen in Figure 6.7.

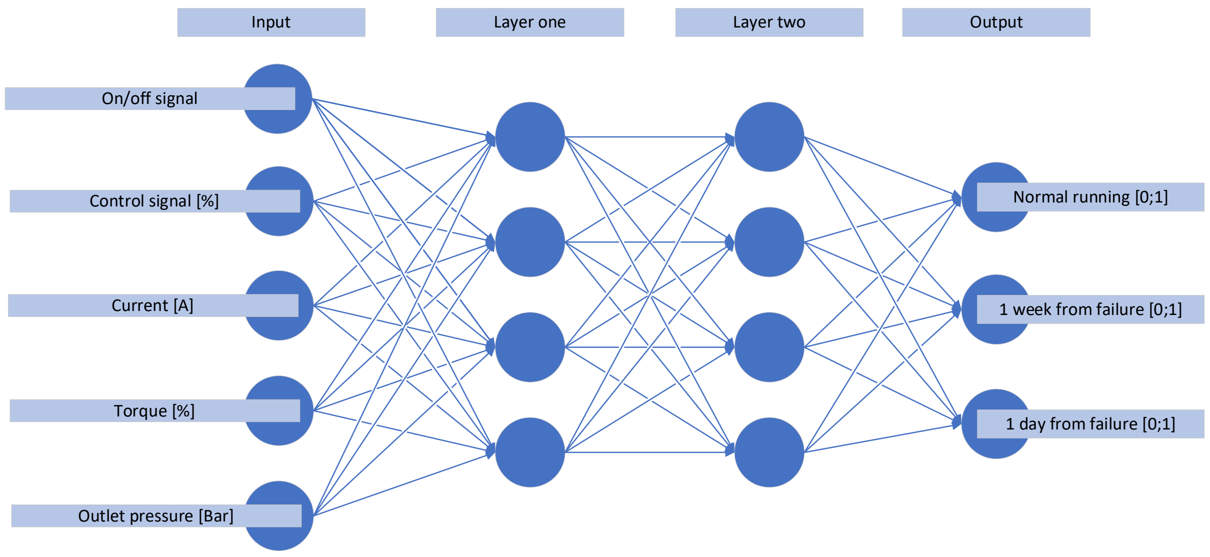


Figure 6.7 Network structure for model describing the states of only pump 19

When the model was tested on pumps that failed after January 2022 from the pump 19 location, the results appear to be close the level of GRU model 7. The model correctly predict *Normal* 100% of the times, *One week from fail* 46.4% and *One day from fail* 0% of the samples. The model will not, from the test, give a false alarm where an alarm is defined as *One week from fail* or *One day from fail* when the state should be *Normal*. The training indicates that the training pipeline is not faulty, although what is modelled and how the structure is built has a large impact on the model results.

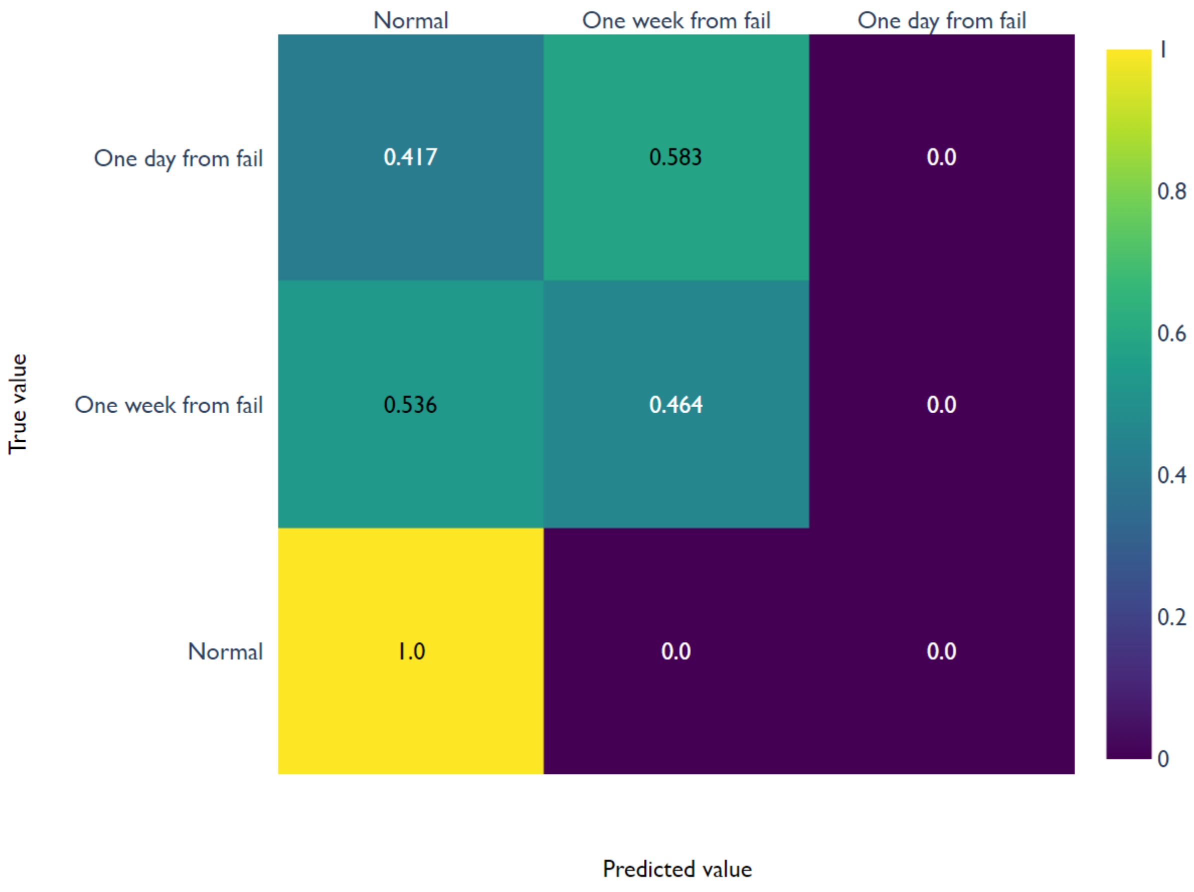


Figure 6.8 Evaluation of LSTM model for only pump 19

6.9 Overfitting

Long short-term memory model 8 introduced the idea that the models may be overfitting as the validation accuracy was constantly high. It may still hold true that the models were overfitting, tools exist to deal with overfitting. Amongst them are "*early stopping, l1 and l2 regularization, dropout, max-norm regularization, and data augmentation*" [34]. It may be useful to investigate such strategies in further studies. These strategies may increase the generality of the model and increase the accuracy of the two critical labels *One week from failure* and *One day from failure*.

7 Conclusion

This thesis has had two main goals; understanding what is needed to set a machine learning in production for predictive maintenance and get useful information from the model. The second part focused on generalizing a model which was initially proved to work for one pump location.

A detailed specification on what is needed to run a machine learning model for predictive maintenance in an industrial plant has been made including a requirement list, infrastructure considerations and a risk assessment.

Many models have been trained using the three machine learning methods; long short-term memory, gated recurrent unit and support vector machine. They have been trained using many different configurations with the goal of being able to model all the pumps in the process part under scrutiny. To get the results, the data was inspected to understand what features may be useful and what pre-processing steps was needed.

The best model had a 100% probability of correctly predicting when the pumps are *normal*, 50% probability of predicting when there is one week to failure, where the remaining 50% was assumed *normal*. It had 31.2% probability of predicting when there is one day before failure, while the remaining 68.8% was predicted as *normal* from the test set that was available.

The results indicate that there will be few false alarms saying that the pump needs maintenance when it does not need it. It may however be conservative in giving alarms that the pump needs maintenance when it does need it.

8 Bibliography

[1] Kepware, "DataLogger," [Online]. Available: https://www.kepware.com/en-us/products/kepserverex/advanced-plug-ins/datalogger/.

[2] Timescale, "TimescaleDB," [Online]. Available: https://www.timescale.com/.

[3] Grafana Labs, "Grafana," Grafana labs, [Online]. Available: https://grafana.com/oss/grafana/.

[4] O. Y. R. A. H. M. Martin Holm, "Machine Learning for predictive maintenance of pumps at "Den Magiske Fabrikken"," 2021.

[5] M. Holm, "An architectural desgin for implementing predictive maintenance in an industrial plant," Not published, 2021.

[6] M. N. K. d. l. M. I. H. A. B. Terry Cox, "MLOps Roadmap 2021," 06 11 2021. [Online]. Available: https://github.com/cdfoundation/sig-mlops/blob/master/roadmap/2021/MLOpsRoadmap2021.md. [Accessed 08 02 2022].

[7] A. K. I. B. A. K. M. P. Larysa Visengeriyeva, "Machine Learning Operations," INNOQ, [Online]. Available: https://ml-ops.org/. [Accessed 08 02 2022].

[8] I. M. W. L. Z. Lenn Bass, DevOps: A Software Architect's Perspective, Addison-Wesley Professional, 2015.

[9] A. K. I. B. A. K. M. P. Dr. Larysa Visengeriyeva, "MLOps Stack Canvas," INNOQ, [Online]. Available: https://ml-ops.org/content/mlops-stack-canvas. [Accessed 10 02 2022].

[10] N.-O. Skeie, Software Engineering Object-oriented Analysis, Design, and Programming using UML and C#, Porsgrunn: Lecture notes, 2019.

[11] Wikipedia, "Unified Process," 11 09 2021. [Online]. Available: https://en.wikipedia.org/wiki/Unified_Process. [Accessed 06 02 2022].

[12] R. K. S. J. K. B. R. S. J. S. Klaus Greff, "LSTM: A Search Space Odyssey," *TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS,* 2015.

[13] P. Remy, "Stateful LSTM in Keras," 30 07 2016. [Online]. Available: http://philipperemy.github.io/keras-stateful-lstm/. [Accessed 16 03 2022].

[14] Wikipedia, "Recurrent neural networks," Wikipedia, 16 02 2022. [Online]. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network#Fully_recurrent. [Accessed 30 03 2022].

[15] C. Olah, "Understanding LSTM Networks," 27 08 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed 14 03 2022].

[16] C. G. K. C. Y. B. Junyoung Chung, "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling," 12 2014. [Online]. Available: https://arxiv.org/pdf/1412.3555v1.pdf. [Accessed 03 17 2022].

[17] Z. C. L. M. L. a. A. J. S. Aston Zhang, "Gated Recurrent Units (GRU)," in *Dive into Deep Learning*, 2022, pp. 346-349.

[18] S. Kostadinov, "Understanding GRU Networks," 16 12 2017. [Online]. Available: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be. [Accessed 17 03 2022].

[19] A. Géron, "Fine-Tuning Neural Network Hyperparameters," in *Hands-On Machine Learning with Scikit-Learn*, Sebastopol, O'Reilly Media, Inc., 2017, pp. 362-364.

[20] K. Eckhardt, "Choosing the right Hyperparameters for a simple LSTM using Keras," Towardsdatascience, 29 11 2018. [Online]. Available: https://towardsdatascience.com/choosing-the-right-hyperparameters-for-a-simple-lstm-using-keras-f8e9ed76f046. [Accessed 15 03 2022].

[21] V. V. Corinna Cortes, "Support-Vector Networks," *Machine Learning,* pp. 273-297, 1995.

[22] F. a. V. G. a. G. A. a. M. V. Pedregosa, "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research,* no. 12, pp. 2825--2830, 2011.

[23] M. H. L. M. Y. L. Jie M. Zhang, "Machine Learning Testing: Survey, Landscapes and Horizons," 21 12 2019. [Online]. Available: https://arxiv.org/pdf/1906.10742.pdf. [Accessed 15 02 2022].

[24] The Linux Foundation, "ONNX," 2019. [Online]. Available: https://onnx.ai/. [Accessed 22 02 2022].

[25] A. K. I. B. A. K. M. P. Dr. Larysa Visengeriyeva, "Three Levels of ML Software," INNOQ, [Online]. Available: https://ml-ops.org/content/three-levels-of-ml-software#code-deployment-pipelines. [Accessed 16 02 2022].

[26] Wikipedia, "High Availability," Wikipedia, 11 02 2022. [Online]. Available: https://en.wikipedia.org/wiki/High_availability. [Accessed 23 02 2022].

[27] Linux Foundation, "LF AI & Data Foundation Interactive Landscape," Linux Foundation, 21 02 2022. [Online]. Available: https://landscape.lfai.foundation/. [Accessed 24 02 2022].

[28] Neptune, "Best End-to-End MLOps Platforms: Leading Machine Learning Platforms That Every Data Scientist Need to Know," 27 12 2021. [Online]. Available: https://neptune.ai/blog/end-to-end-mlops-platforms. [Accessed 24 02 2022].

[29] S. H. Marvin Rausand, "Security," in *Risk Assessment: Theory, Methods, and Applications*, John Wiley & Sons, 2020, p. 51.

[30] Wikipedia, "Feature Scaling," 21 09 2021. [Online]. Available: https://en.wikipedia.org/wiki/Feature_scaling.

[31] A. Géron, "Irrelevant Features," in *Hands-On Machine Learning with Scikit-Learn & TensorFlow*, O'Reilly Media, Inc., 2017, p. 49.

[32] F. a. o. Chollet, "Probabilistic losses," 2015. [Online]. Available: https://keras.io/api/losses/probabilistic_losses/.

[33] A. Géron, "Handling Text and Categorical Attributes," in *Hands-On Machine Learning with Scikit-Learn*, Gravenstein Highway North, Sebastopol, O'Reilly Media, Inc., 2017, pp. 92-93.

[34] A. Géron, "Avoiding Overfitting Through Regularization," in *Hands-On Machine Learning with Scikit-Learn*, Sebastopol, O'Reilly Media, Inc., 2017, pp. 406-415.

[35] Wikipedia, "Hadamard product," Wikipedia, 22 03 2022. [Online]. Available: https://en.wikipedia.org/wiki/Hadamard_product_(matrices). [Accessed 30 03 2022].

Appendices

Appendix A. Task description

FMH606 Master's Thesis

Title: Specifying a machine learning operational framework, refinement and scaling of machine learning models for progressive cavity pumps at Den Magiske Fabrikken

HSN supervisors: Carlos F. Pfeiffer. Carlos.Pfeiffer@usn.no, Håkon Viumdal. Hakon.Viumdal@usn.no

External partner: Lindum, Drammen. Contact: Frode Steen. Frode.Steen@lindum.no

Task background:

Lindum produces biogas from food waste and animal manure at "Den Magiske Fabrikken" (DMF). In one segment of the process, there are 8 pumps which often are replaced due to being damaged. The factory is aiming to reduce the costs of these high frequent replacements, by using a predictive maintenance approach. The gained health prognosis will help deciding when the pumps should be overhauled, and reducing the number of total breakdowns. This has the prospective of buying less new pumps, but more spare parts.

Task description:

Previous projects have been done on finding estimates of progressive cavity pump states in the sense of how damaged the pumps are, purely by considering time from failure, and to identify some of the needs for operation. These findings should be used to create models for all the pumps and prepare an implementation of the models into a system which can monitor the physical pump. The process data can be accessed through OPC UA for near real-time data or a PostgreSQL database for historical data. The results needs to be conveyed to the operators/manager at the factory.

The main goal of this master thesis is to define a specification for implementation and refine and scale the models. The expected tasks are:

- Complete a literature review on machine learning model implementation/infrastructure (MLOps) and machine learning methods
- Create specifications for the software
- Additional learning
  - Reinforce model with new data
  - Try out new methods and features

    - Test models offline
    - Scale model to other pumps
- Investigate how to communicate with the SCADA system
- Investigate how to communicate with the maintenance system
- Implementation, if the factory management allows for it
- Complete and deliver the master thesis report
- Prepare and deliver a thesis presentation

Student category: IIA

This project is reserved for the Industry master student working at Lindum

Practical arrangements:

The student will have access to the factory, and the related SCADA system. He has also got the necessary support from the factory management to run this project.

Signatures:

Student (date and signature):

01.02.2022 Martin Holm

Supervisors (date and signature):

01.02.2022 [illegible] [illegible]

01.02.2022 [illegible]

Appendix B. Predictive Maintenance of pumps at ‘Den Magiske Fabrikken’, using Machine Learning Technique

Predictive Maintenance of pumps at ‘Den Magiske Fabrikken’, using Machine Learning Techniques

Martin Holm[1,2], Ozgur Yalcin[1], Carlos Pfeiffer[1], Håkon Viumdal[1]

[1] Faculty of Technology, Natural Sciences, and Maritime Studies, University of South-Eastern Norway, Norway,
[2]Lindum AS, Norway, martin.holm@lindum.no
ozgurylc@gmail.com, {carlos.pfeiffer, hakon.viumdal}@usn.no

Abstract

In this work, we investigate machine learning methods to predict the failures of progressive cavity pumps (PCP). The PCPs are located in a biogas plant, Den Magiske Fabrikken, in Norway, which is transforming food waste and animal manure to biogas and biofertilizer. Available measurements were pump on-signal, speed, current, torque and control signal, inlet flow, inlet pressure and outlet pressure, and several vibrations derived signals.

Five categories were defined to categorize the operation of the pumps as: “stopped”, “normal running”, “7 days from failure”, “1 day from failure” and “1 hour from failure”. The objective was to train a Machine Learning model to predict these categories. The data was pre-processed to clean gross outliers and scale the signals using different techniques.

This paper presents results from the same Long Short-Term Memory (LSTM) model using two different approaches for scaling the data. The results are evaluated using confusion matrices where one scaling method clearly improves the results when testing on new data points.

Further work is presently being carried out to implement the selected methods in real-time and to generalize the model.

Keywords: Machine Learning, Predictive Maintenance, Long-Short Term Memory, Progressive Cavity Pump

1 Introduction

This project investigates and evaluates progressive cavity pump failures used in a waste processing plant by applying Long Short-Term Memory (LSTM) machine learning methodology. In the industry, maintenance costs account for significant losses in profit for companies. Predictive maintenance methods and their tools have changed the way in how to approach problems through advanced control analysis.

Lindum operates a waste management company that produces biogas and biofertilizer as a result of processing animal manure and food waste. During the processing phase, the highly corrosive and acidic liquid flows through the pipes and causes severe effects on the pumps. To prevent any production losses and increase the pump lifetime, pumps are maintained periodically. Obviously, excessive manual supervision of the pumps may result in increased labor force demand and increase the spare part costs. However, increased periodic surveillance do not prevent unexpected failures completely. Consequently, the objective of this project is to develop methods for preventing pump failures by analyzing pump parameters with ML algorithms and proposing a model to detect faults.

1.1 Progressive Cavity Pumps and Predictive Maintenance

Positive displacement pumps can handle solids, high viscosity and low flow rates. Besides, progressive cavity pumps are one type of positive displacement pump. Centrifugal pumps, on the other hand, are suitable for low viscosity and high flow rates. The pump efficiency will decrease at both higher and lower pressures for centrifugal pumps, whereas the pump efficiency will increase with increasing pressure in positive displacement pumps.

In this project, the analyzed pump type is 'Nemo' brand progressive cavity pumps produced by Netzsch Pumpen & Systeme GmbH. These types of pumps provide a large capacity and pressure range. During the operation of the process in the factory, the pumps suffer from changing viscosity and corrosive materials in each batch. Figure 1.0 illustrates the progressive cavity pump that is used in the process. The pump has the following components: rotor (1), stator (2a, 2b), drive chain (3), shaft sealing (4), suction and discharge housing (5). Typical problems in progressive cavity pumps are elastomer expansion, rotor, and stator material corrosion which are caused by high temperature or fluid type (Lea et al., 2003).

There are some points to avoid pump failure specifically in progressive cavity pumps. These are:

- Choosing the right elastomer type by taking into account temperature and fluid physical properties
- Avoiding dry running conditions
- Selecting suitable rotor material to stay away from abrasive wear on the rotor

Predictive maintenance aims to transform advanced analytical and process data into valued outcomes. Hence, equipment failure or breakdown can be prevented just before it occurs. Additionally, predictive maintenance may take advantage of machine learning (ML) algorithms to build a systematic approach. Besides, predictive maintenance minimizes the cost of maintenance and improves the equipment lifetime without causing unpredicted production losses. Thus, the process will run as long as possible without interruption.

1.2 Machine Learning Methods

Various type of data is gathered from the process equipment. ML algorithms are able to unveil unseen or hidden patterns and relationships within a data set. With the progressively increase of computational power, and development of new ML algorithms, there is an increasing trend in publications in the literature related to data analysis through ML algorithms (Carvalho et al., 2019). One method is the LSTM algorithm, which is considered especially successful in time series applications, where long-term dependencies in the data needs to be detected (A. Géron, 2019, pp.511-523). Simply, the function stores a value and determines how long it should be stored. This makes long short - term memory one of the most common models when working with time-dependent data (Rivas et al.,2019). Wisyaldin (2020) compared Autoregressive Moving Average (ARMA), Recurrent Neural Network (RNN), and LSTM models for analyzing vibration signals to predict the health condition of

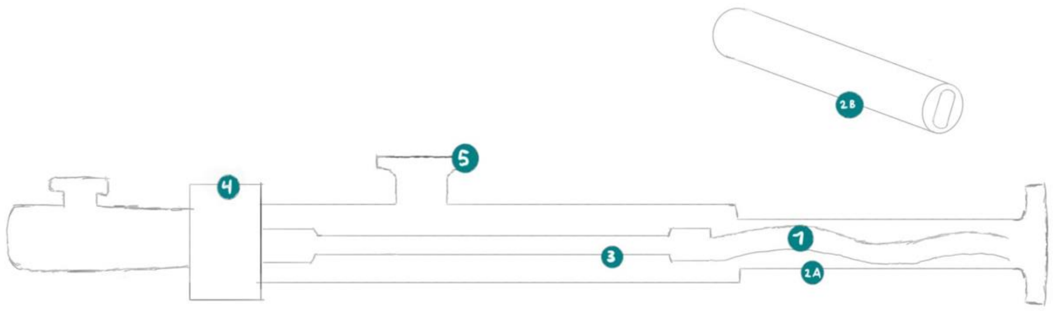


Figure 1.1 Illustration of the progressive cavity pump that is used in the process.

bearings of a water circulation pump and LSTM produced better accuracy. Even though LSTM is used to calculate remaining useful time and anomaly detection in various processes, there are few studies for progressive cavity pump failure analysis with LSTM found in the literature.

2 System Description

2.1 Features

The system under scrutiny in this paper consists of a progressive cavity pump with measurements control signal [%], current [A], torque [%] and speed [%] from a frequency converter. In addition, inlet pressure [Bar], outlet pressure [Bar] and inlet flow [m³/h] is measured. These will be used as the features for the machine learning model. The sampling rate for all the measurements is 30 seconds. Although the selected pump is part of a system of pumps and may be impacted by other pumps earlier in the process, this potential impact has been ignored in this work. The system cyclically pumps fluid for 45-60 minutes, it will always start the cycle again after 60 minutes whether it has just ended or ended 15minutes ago.

The analyzed feature data spans 17 months, with some missing data. During this period, the pump considered has been replaced 14 times due to pump failures.

2.2 Predictions

The goal is to predict one of the five operational categories: pump is (0) stopped, (1) running normally, (2) less than one week from failure, (3) less than one day from failure or (4) less than one hour from failure. Where running normally is assumed to be anything which is not covered by the other categories.

These categories have been assumed useful as there was little information concerning the breakdown of the pumps, only sparse information about when they had been replaced was available.

3 Methods and Methodology

3.1 Long Short-Term Memory Configuration

The LSTM model architecture was set up as a two layered LSTM block with a dense output layer as seen in Figure 3.1. The first layer has 7 feature inputs with a sequence length of 120 and 32 output neurons. The layer has the parameter return_sequence set as true (Chollet, LSTM layer, 2015) which means a sequence will be returned, compared to only return the last estimate of the sequence, which is the case when set to false. The sequence length of 120 samples correspond to one hour which is the cycle time for the pump sequence. The pump sequence is determined by the process operation. The 32 neurons from the first layer serves as inputs to the second layer. However, it outputs only 16 neurons as the return sequence is set to false. Both the LSTM layers are using standard configurations for all other parameters. Lastly, a dense layer using the softmax activation function with the 16 neurons from the previous layer as inputs and outputting a probability for each of the categories. The output with the highest probability is assumed to be correct for a given sequence thus giving a positive for one of the five categories. The model is compiled using the loss function categorical crossentropy (Chollet, 2015) and using the Adam optimizer (Kingma, 2017).

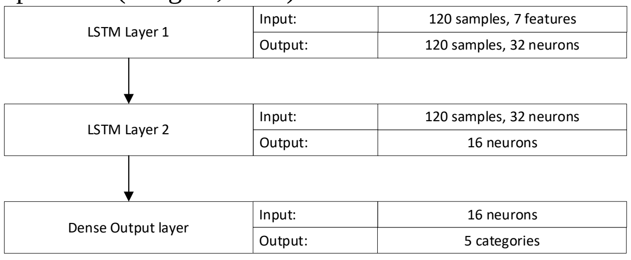


Figure 3.1 LSTM model architecture

3.2 Scaling

Standardization is used to scale the data, using eq. (1) where z is the scaled sample, x is the sample that should be scaled, μ is the mean and σ is the standard deviation.

$$z = \frac{x - \mu}{\sigma} \quad (1)$$

The standardization is used in two ways, one where the entire dataset is scaled using the same scaler for the entire dataset. The other approach is to collect data during one hour of operation for each pump and use this data to calculate individual means and standard deviations to scale the new data. Both approaches are shown in Figure 3.2.

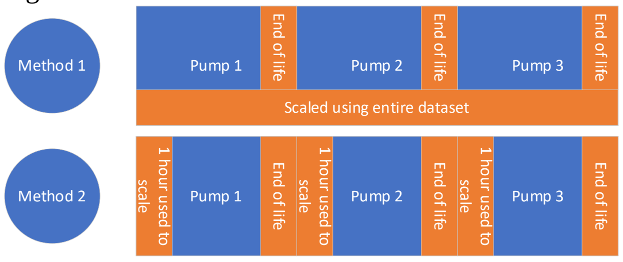


Figure 3.2 Two methods to scale training data

The reasoning behind this approach can be seen by inspecting parts of the data shown in Figure 3.3. There are several normal distribution-like structures in the current when including data from all the pumps in the same histogram. This may indicate that the level of current (and other variables) may vary between the pumps that have been replaced, and thus the level near the end of

the lifetime may vary. This gives the ML method ambiguous signals as to what is considered a degraded pump.
Looking at one single physical pump scaled with the first hour of its own data, the distribution seems closer to a single normal distribution as depicted in Figure 3.4, yet it has two distinct tops. Investigating other plots reveals that many look like Figure 3.4 and some are a lot closer to a narrow normal distribution.

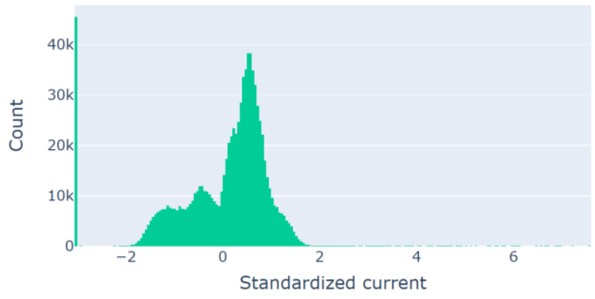


Figure 3.3 Histogram including all pump failures with previous original scaling method for current.

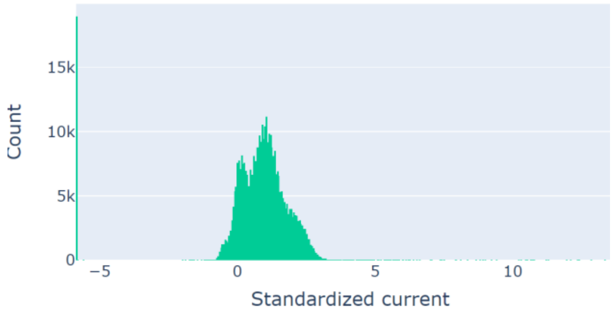


Figure 3.4 Histogram for one pump failure for the feature current, scaled.

It thus appears that there are individual characteristics of the pumps, and hence they have various distributions. This may confuse the LSTM-model as there will be many levels of data points where the pump is ok for one pump, but not for another.

Continuing this approach and using data from only the first hour of the pump's active lifetime yields a distribution as seen in Figure 3.5. This distribution looks more coherent, yet there are more outliers and a higher span.

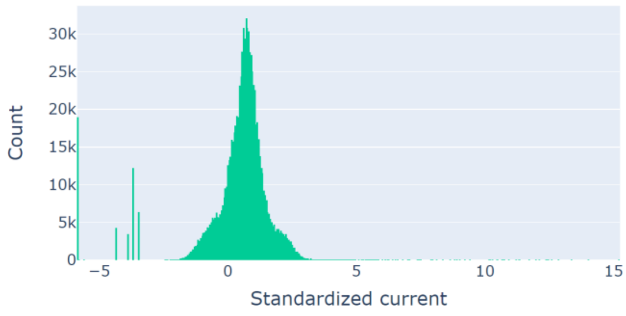


Figure 3.5 Histogram including all failures with scaling method 2.

3.3 One hot encoding

The outputs are one hot encoded from integer encoded, meaning that the labels have been converted to numbers as seen in Table 4. These in turn has been transformed into a one hot encoded format, where each row indicates an example where only one label is true, and others are false. As per definition of one hot encoding (Géron, 2017).

Table 4. Integer encoded labels

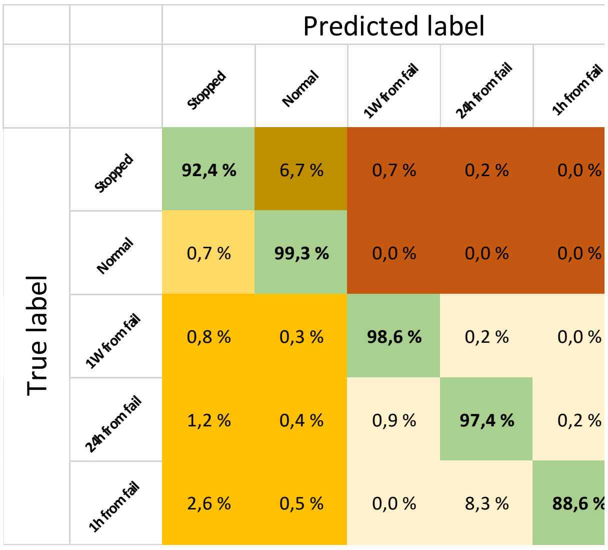
Label	Description
0	Stopped
1	Normal running
2	Less than one week before failure
3	Less than 24 hours before failure
4	Less than 1 hour before failure

4 Results and Discussion

4.1 Scaling pump data using method one

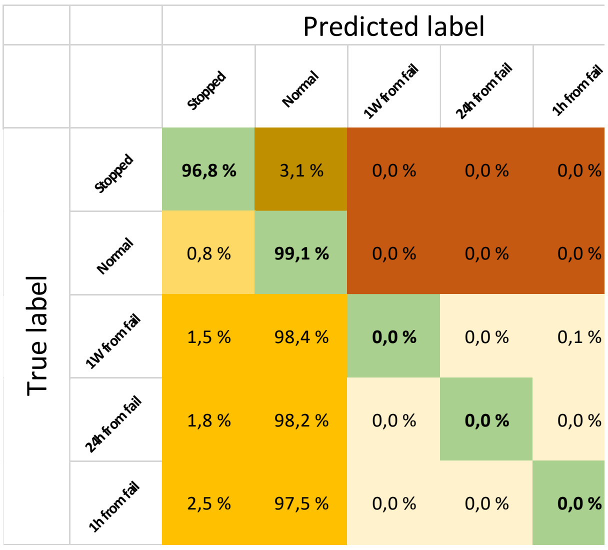
Before the data is used for training, the features are standardized. All the 14 failures are scaled using one scaler and the data is split into sequences. The outputs are one hot encoded. After this, the data is split into training and testing datasets with 60% used for training, 20% for validation and 20% used for testing.

Using this method, the results on the confusion matrix based on the training set can be seen in Figure 4.1 and appears very good. Figure 4.2 however shows the results in a more realistic manner where the data tested on was not involved in training the model. The model was trained on data from March 2020 to September 2021 and was then tested on data from September 2021 to October 2021. The confusion matrix for the test data shows that all the three categories where it was less than one week before failure of the pump, was considered “normal operation”, or in some rare cases “stopped”. Some of the reason for this might be related to an extensive number of “normal operation” in the data, compared to the other. That will affect the model. Comparing Figure 4.1 and Figure 4.2, there is a clear indication of a generalization problem with the model. As already mentioned, the scaling method 1 has its short-comings, which is improved in method 2.



		Predicted label				
		Stopped	Normal	1W from fail	24h from fail	1h from fail
True label	Stopped	**92,4 %**	6,7 %	0,7 %	0,2 %	0,0 %
	Normal	0,7 %	**99,3 %**	0,0 %	0,0 %	0,0 %
	1W from fail	0,8 %	0,3 %	**98,6 %**	0,2 %	0,0 %
	24h from fail	1,2 %	0,4 %	0,9 %	**97,4 %**	0,2 %
	1h from fail	2,6 %	0,5 %	0,0 %	8,3 %	**88,6 %**

Figure 4.1 Confusion matrix based on training data



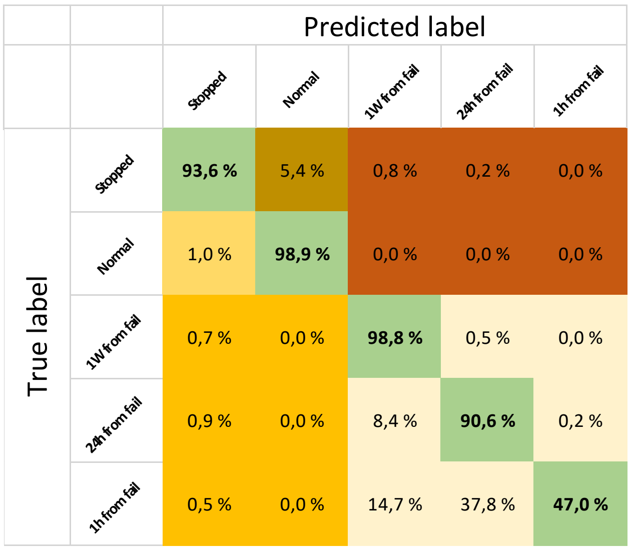
		Predicted label				
		Stopped	Normal	1W from fail	24h from fail	1h from fail
True label	Stopped	**96,8 %**	3,1 %	0,0 %	0,0 %	0,0 %
	Normal	0,8 %	**99,1 %**	0,0 %	0,0 %	0,0 %
	1W from fail	1,5 %	98,4 %	**0,0 %**	0,0 %	0,1 %
	24h from fail	1,8 %	98,2 %	0,0 %	**0,0 %**	0,0 %
	1h from fail	2,5 %	97,5 %	0,0 %	0,0 %	**0,0 %**

Figure 4.2 Confusion matrix based on test data

4.2 Scaling pump data using method 2

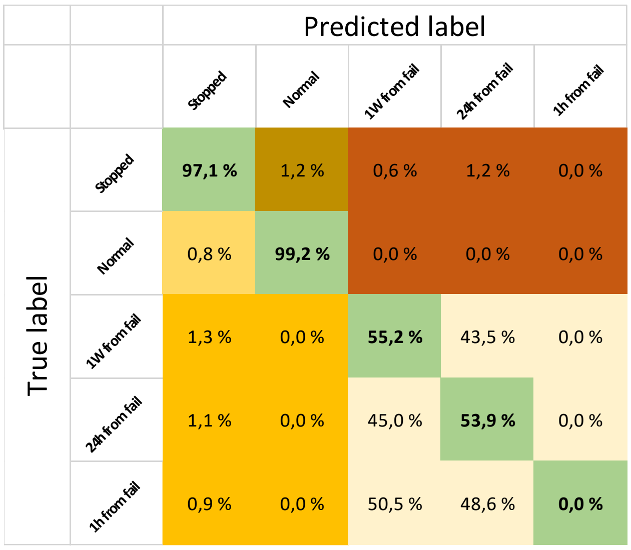
Using the new method for scaling the data has reduced the overall accuracy of the model based on the training data, as seen in Figure 4.3. It can however be noted that most false positives in the failure categories for the most part end up in another failure category.

The test set shows that the model is greatly improved by using the new scaling method in Figure 4.4. The total accuracy becomes 78.5% where the total accuracy is defined by how many samples are correct for each label divided by number of samples tested on.



		Predicted label				
		Stopped	Normal	1W from fail	24h from fail	1h from fail
True label	Stopped	**93,6 %**	5,4 %	0,8 %	0,2 %	0,0 %
	Normal	1,0 %	**98,9 %**	0,0 %	0,0 %	0,0 %
	1W from fail	0,7 %	0,0 %	**98,8 %**	0,5 %	0,0 %
	24h from fail	0,9 %	0,0 %	8,4 %	**90,6 %**	0,2 %
	1h from fail	0,5 %	0,0 %	14,7 %	37,8 %	**47,0 %**

Figure 4.3 Confusion matrix on test data with new scaling method from training data



		Predicted label				
		Stopped	Normal	1W from fail	24h from fail	1h from fail
True label	Stopped	**97,1 %**	1,2 %	0,6 %	1,2 %	0,0 %
	Normal	0,8 %	**99,2 %**	0,0 %	0,0 %	0,0 %
	1W from fail	1,3 %	0,0 %	**55,2 %**	43,5 %	0,0 %
	24h from fail	1,1 %	0,0 %	45,0 %	**53,9 %**	0,0 %
	1h from fail	0,9 %	0,0 %	50,5 %	48,6 %	**0,0 %**

Figure 4.4 Confusion matrix on test data completely separate from training data with new scaling method

The expectation to be able to predict one hour before failure may have been high, however, the other failure categories appear to be reasonable, at least a combination of them. Assuming all failure modes are merged to one, the total accuracy of the model becomes 98.9%. One week from fail: 98.7%, One day from fail: 98.9%, and one hour from fail: 99.1% when adding together each row.

However, the model does never predict a normal operation as an upcoming failure for the given test set. The results indicate that there is a small risk of having a false alarm, and performing pump replacements without any good reason, with this approach. Simultaneously, there is a good chance of being able to detect an approaching error within a pump in advance. On the other side, it is a risk for not being able to detect precisely when the pump failures will occur.

The model was only tested on one pump failure, while trained on many. As such there may be a lucky draw that the model was able to predict as well as it did. It has already been seen that the data varies from pump to pump, and it may be that other pump failures are not that well picked up.

In the process of LSTM modelling, the input data was not filtered out of the noise and raw data was fed into the model. One might argue that noise filtering can increase accuracy. However, it was concluded that noise in data still can hold valuable information, and disregarding noise in data might reduce the model performance.

5 Conclusions and Further Work

This paper has aimed at evaluating and predicting of progressive cavity pump failures in the waste processing plant, Lindum AS. After gathering information from field instruments, measurement data classified 5 different pump working time cycles such as stopped or normal condition, 1h, 24h, and 1 week from failure. Analyzed data covered 17 months of operation that consist of 14 replacements, and with 30s sampling rate. The time series data was handled well by the LSTM algorithm, and produced reasonable results. However, it became evident that scaling for the entire dataset led to information loss on pump failures. Instead, improved results were obtained by scaling pump data with one hour of operational data for each pump replacement. Thus, each pump data was captured on the scaled dataset, separately. The total accuracy of the model with the proposed scaling method becomes 78.5%.

Further work is being done on trying to generalize the model such as to fit onto similar pumps in the process. This requires some features to be removed and use data from many more pumps.

As the stopped label is already known, it is not really needed to predict and will be removed in further studies. The one hour from failure label is never predicted outside the training data and is thus removed from further studies.

More work should be done on setting correct parameters of the LSTM structure.

The initial project also explored other ML techniques such as Support Vector Machine, Naïve Bayes and Principle Component Analysis. These were not tested with the new scaling method and may be worthwhile to investigate further.

Acknowledgements

We acknowledge the collaboration with and support Lindum AS for providing data. Credits for Ronnie Andre Horne Moe (ronnie.moe@boliden.com), who was in the project team, for building LSTM architecture and coding support throughout the project.

References

Thyago P. Carvalho, Fabrízzio A. A. M. N. Soares, Roberto Vita, and João P. Basto. A systematic literature review of machine learning methods applied to predictive maintenance, *Computers & Industrial Engineering*, 137(106024), 2019, ISSN 0360-8352, doi: 10.1016/j.cie. 2019.106024.

François Chollet. LSTM layer, 2015. Retrieved from https://keras.io/api/ layers/recurrent_layers/lstm/.

François Chollet. A. Probabilistic losses, 2015. Retrieved from https://keras.io/api/ losses/probabilistic_losses/.

Aurélien Géron. Handling Text and Categorical Attributes. In *Hands-On Machine Learning with Scikit-Learn*, pages 92-93. O'Reilly Media, Inc, 2021. ISBN: 9781492032649.

Sepp Hochreiter and Jürgen Schmidhuber. Long-Short Term Memory, *Neural Computation*, 9(8):1735 - 1780, 1997, doi: 10.1162 /neco.1997.9.8.1735.

Diederik P. Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. Retrieved from arXiv: https://arxiv.org/abs/1412.6980.

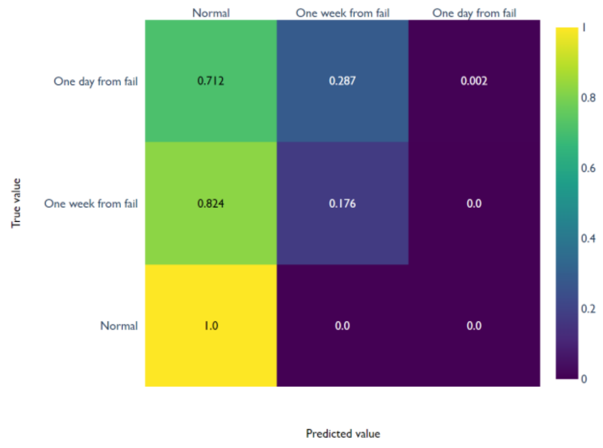
James Lae, Henry V. Nickens, and Mike. R. Wells. *Gas Well Deliquification*, Gulf Professional Publishing, 2003. doi: 10.1016/B978-0-7506-8280-0.X5001-X.

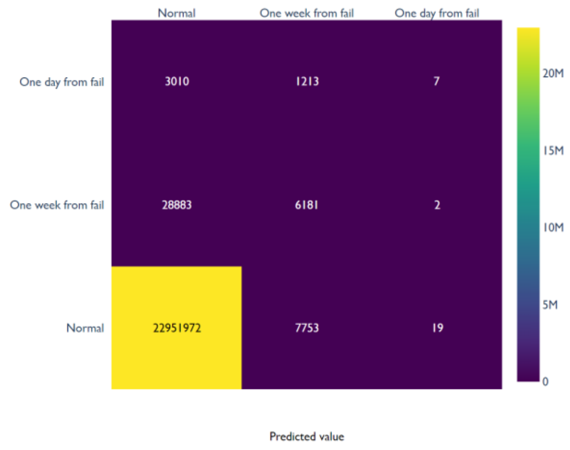
Alberto Rivas, Jesús M. Fraile Pablo Chamoso, Alfonso González-Briones, Inés Sittón, and Juan M. Corchado. A. Predictive Maintenance Model Using Recurrent Neural Networks, In *14th International Conference on Soft Computing Models in Industrial* *and Environmental Applications (SOCO 2019)*, *Advances in Intelligent Systems and Computing*, Springer, pages 261-270, 2019. doi: 10.1007/978-3-030-20055-8_25.

Muhammad Kamal Wisyaldin, Gita Maya Luciana, and Henry Pariaman. Using LSTM Network to Predict Circulating Water Pump Bearing Condition on Coal Fired Power Plant, *2020 International Conference on Technology and Policy in Energy and Electric Power (ICT-PEP)*, pages 54-59, 2020, doi: 10.1109/ICT-PEP50916.2020.9249905.

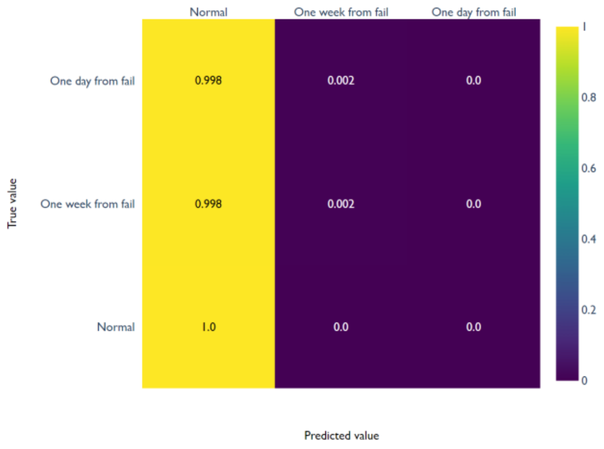
Appendix C. All long short-term memory results in confusion matrices

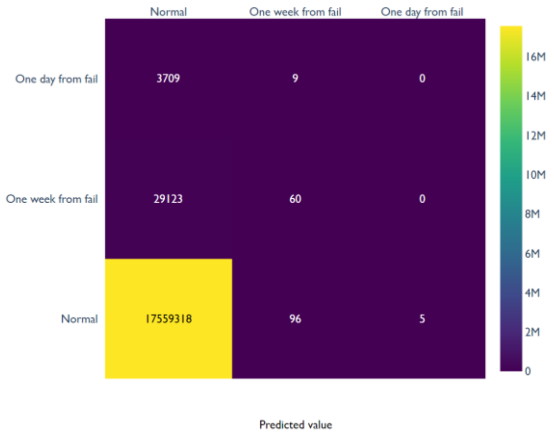
LSTM model 1



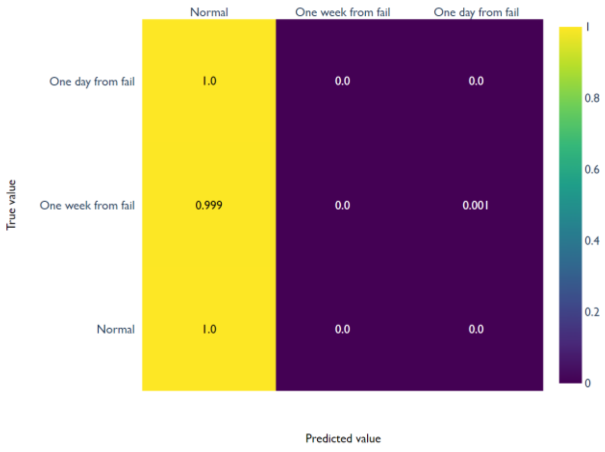


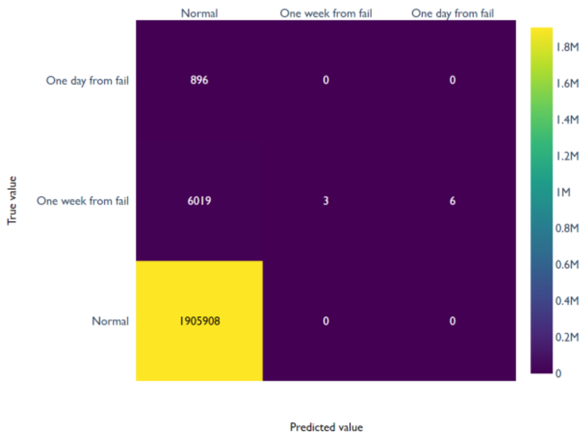
LSTM model 2



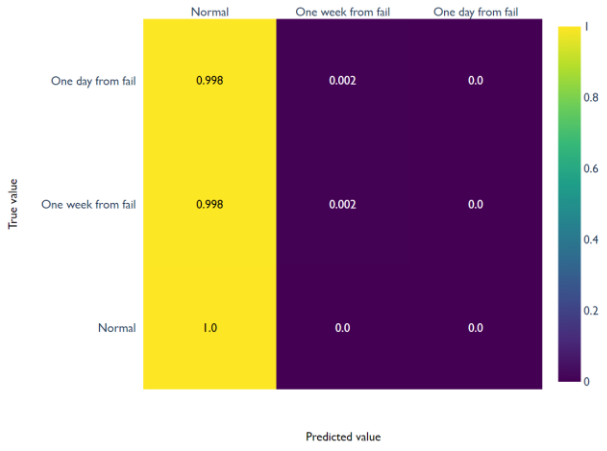


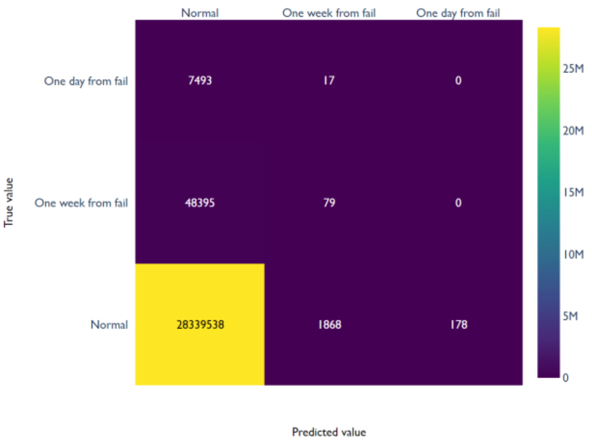
LSTM model 3



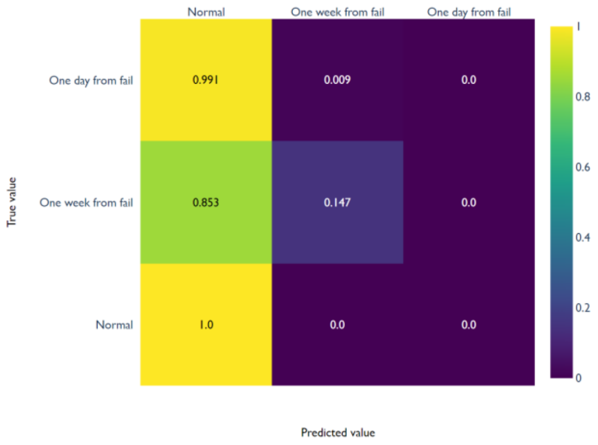


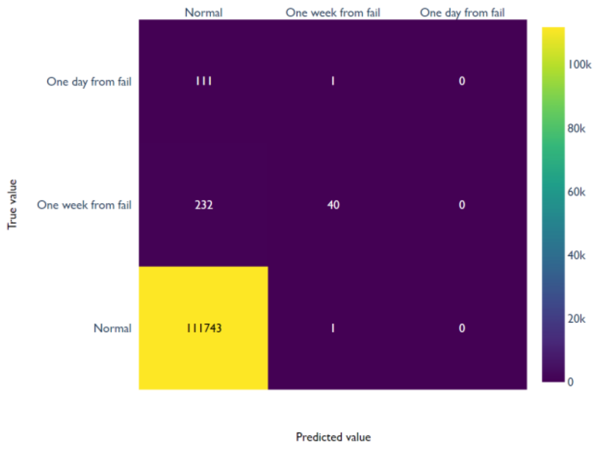
LSTM model 4



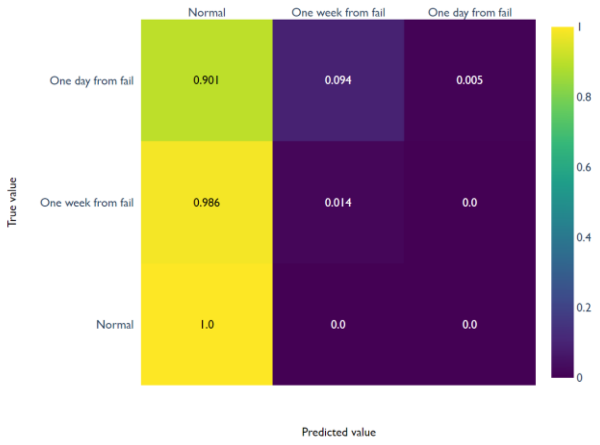


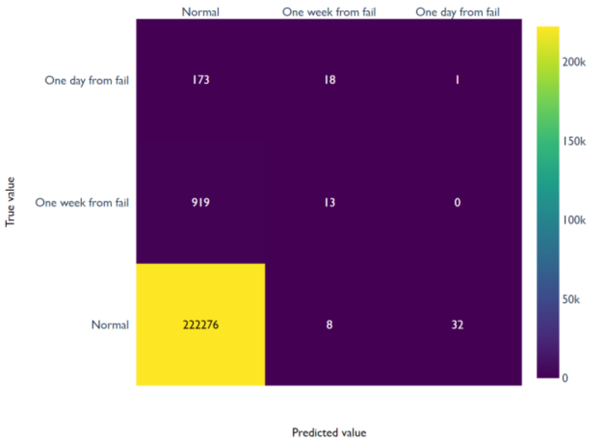
LSTM model 5



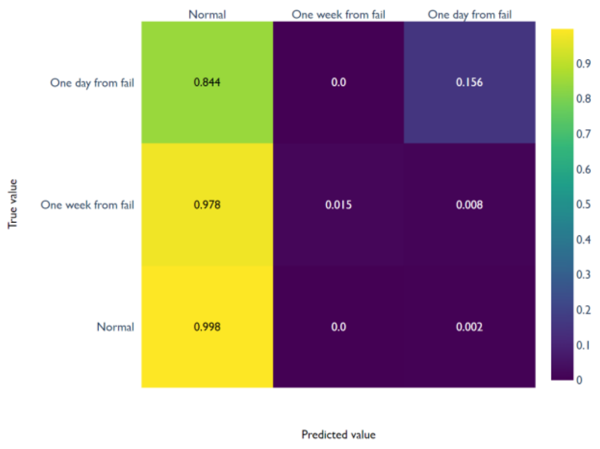


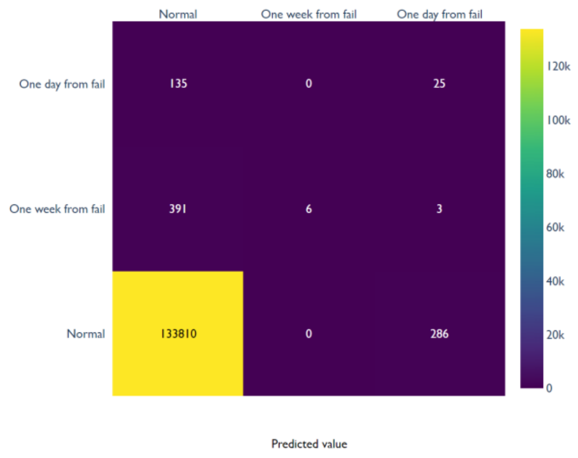
LSMT model 6



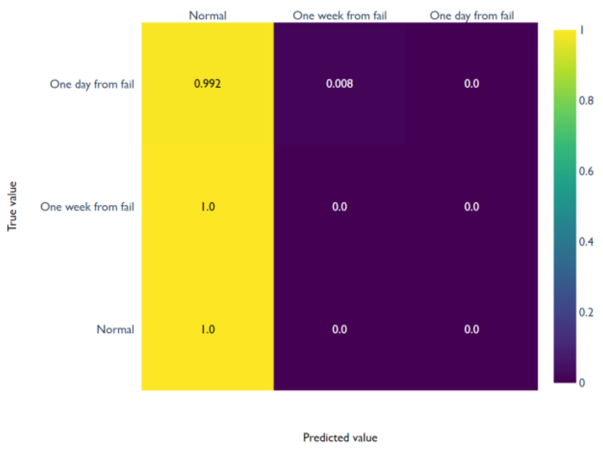


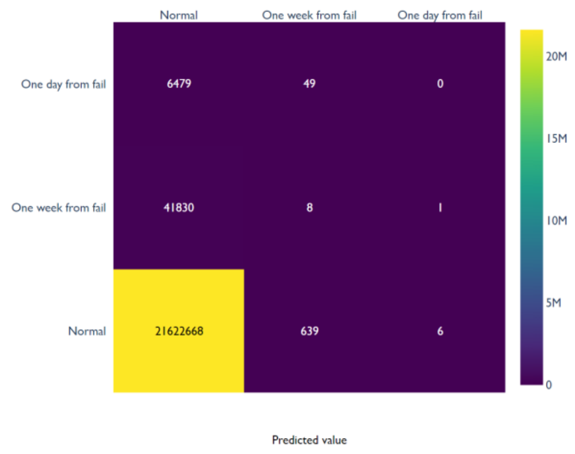
LSTM model 7



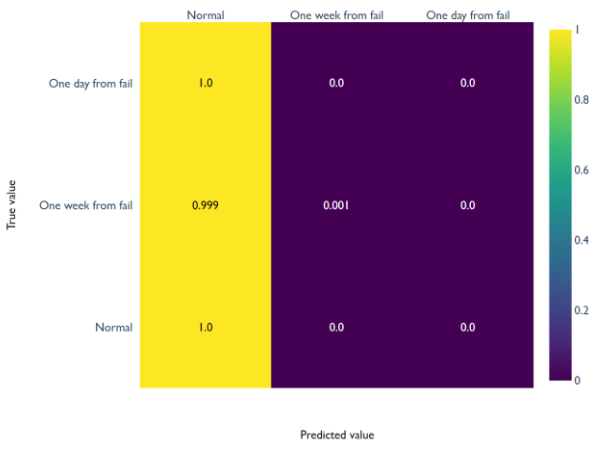


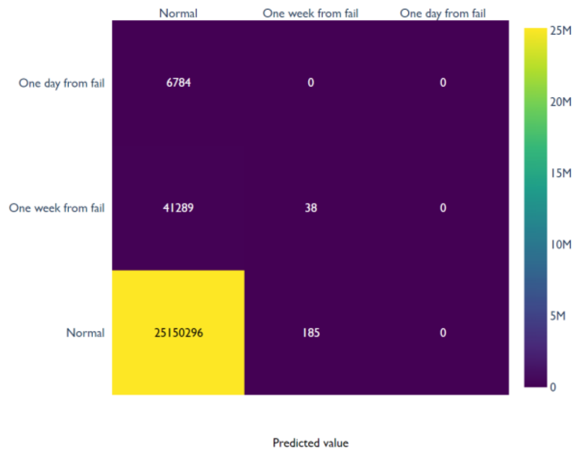
LSTM model 8



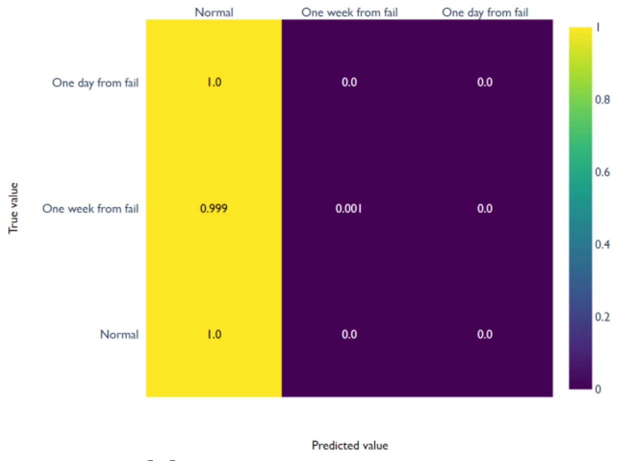


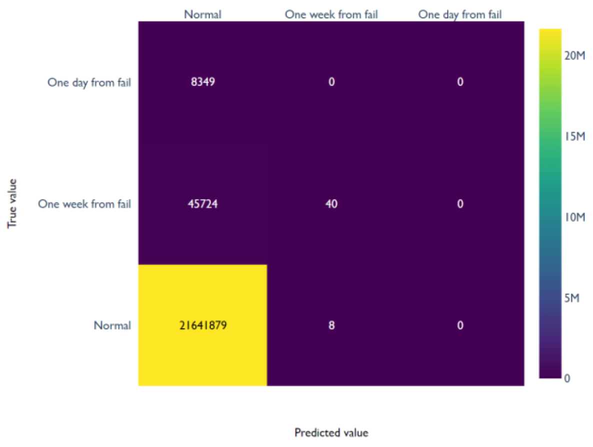
LSTM model 9



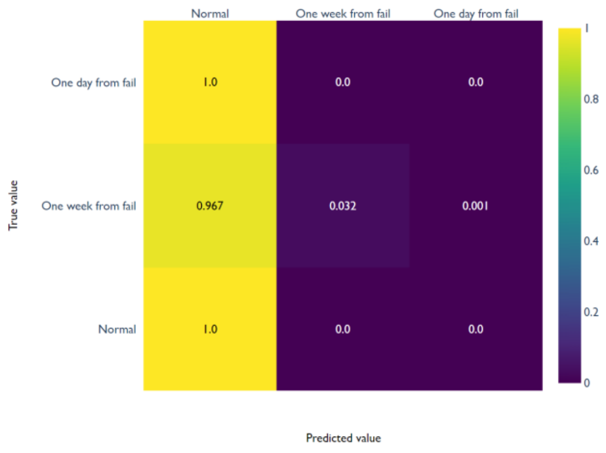


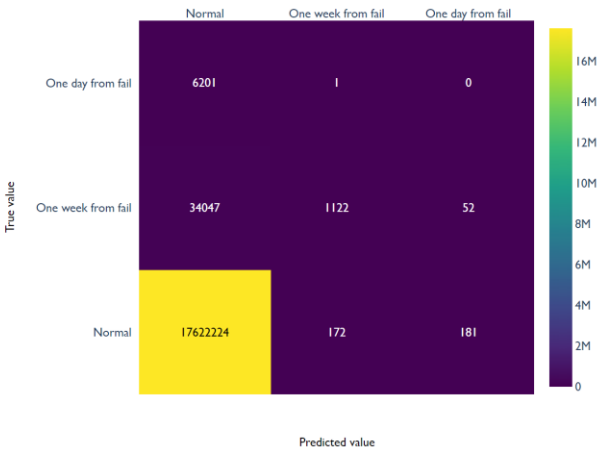
LSTM model 10



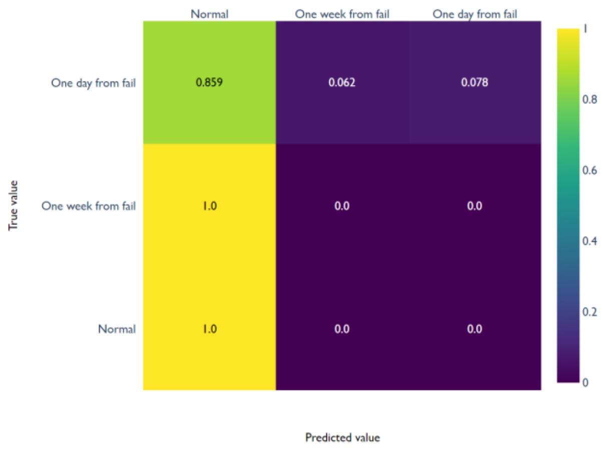


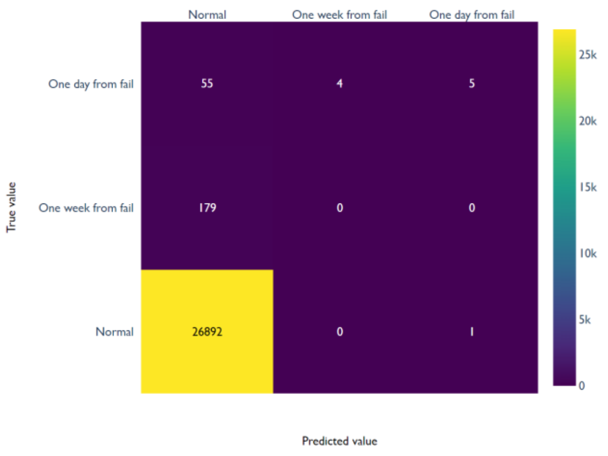
LSTM model 11



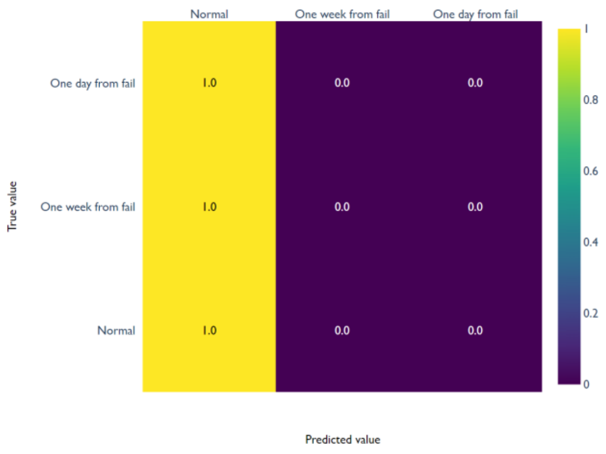


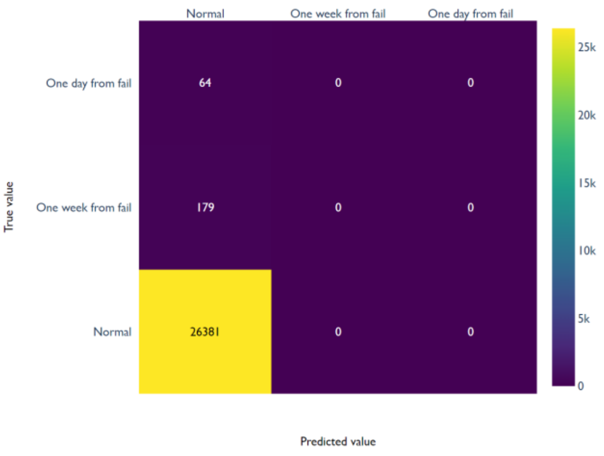
LSTM model 12



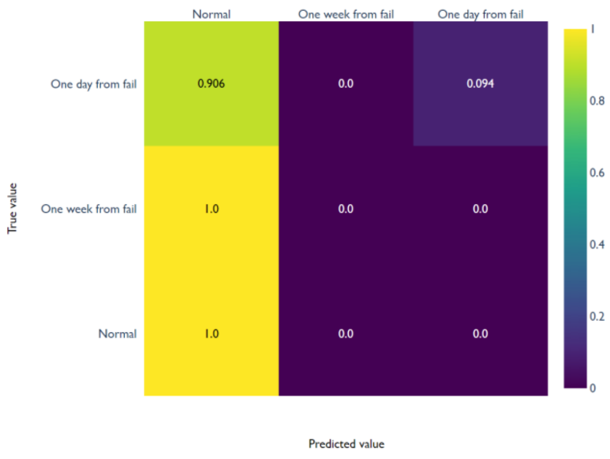


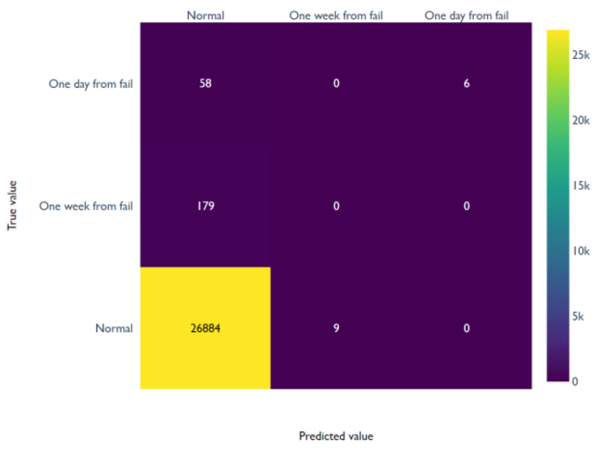
LSTM model 13



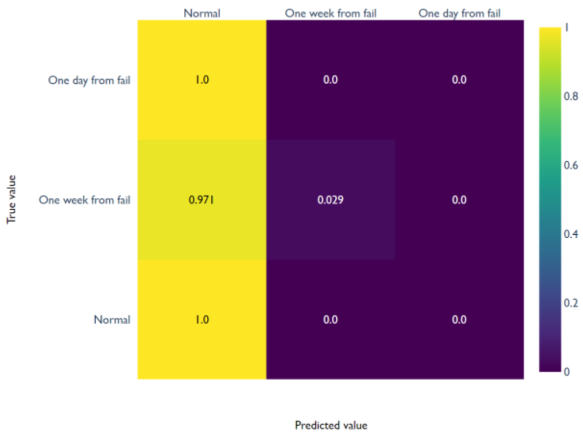


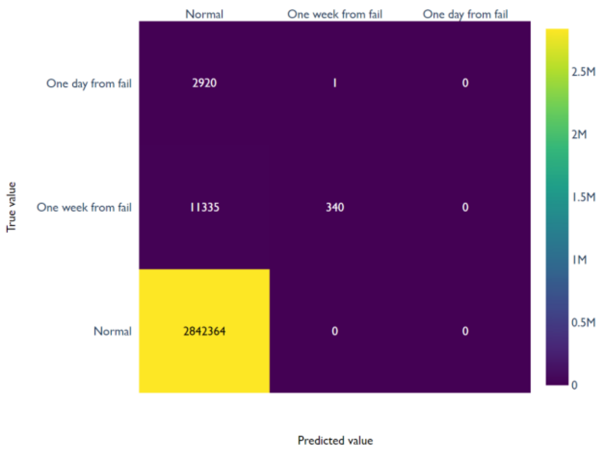
LSTM model 14





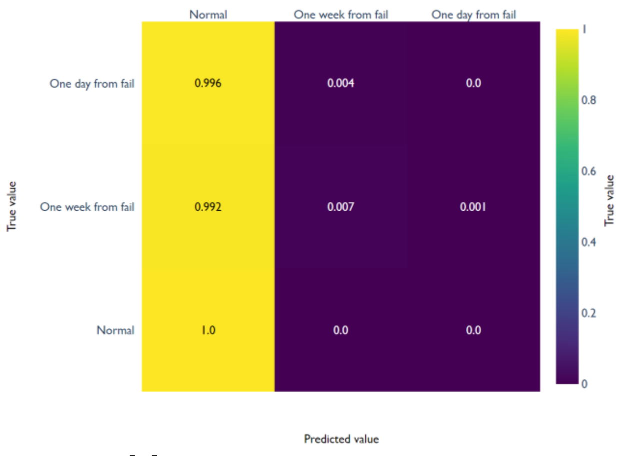
LSTM model 15

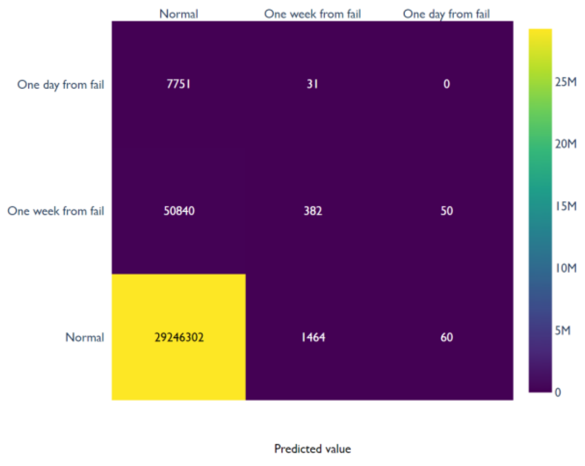




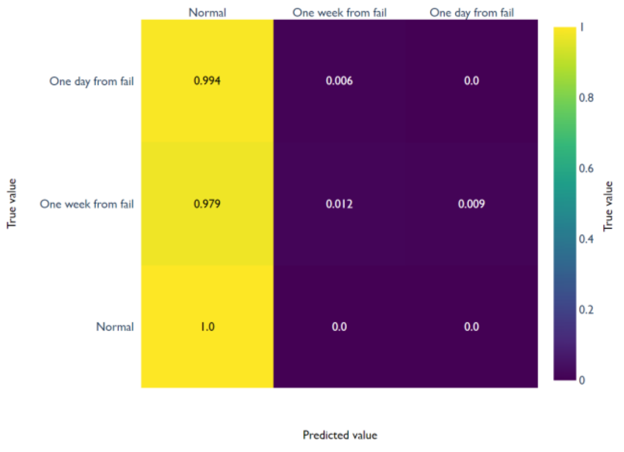
Appendix D. All gated recurrent unit results in confusion matrices

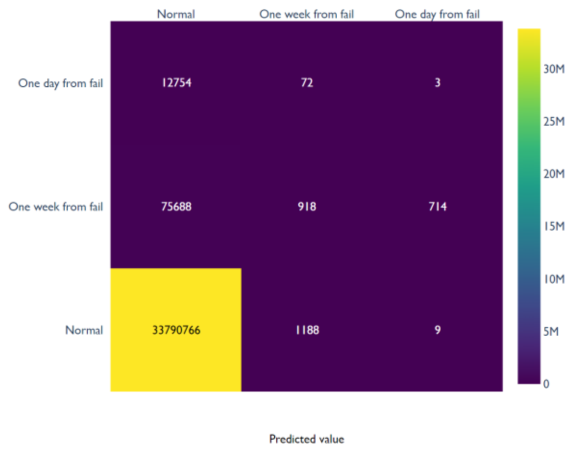
GRU model 1



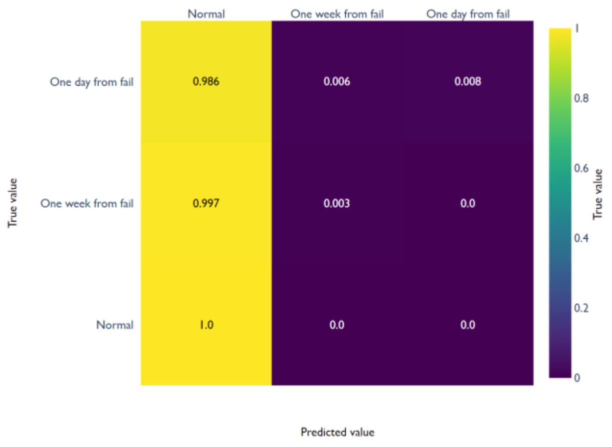


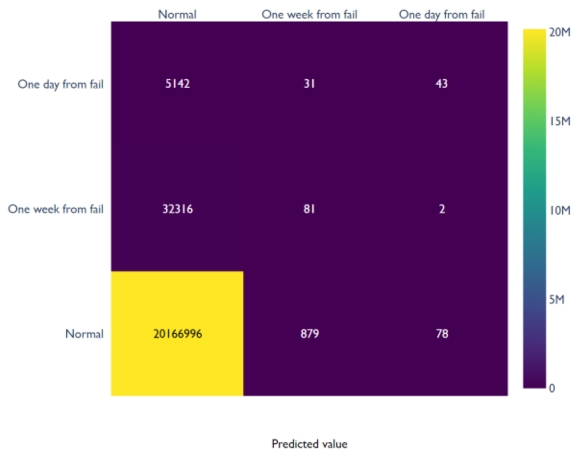
GRU model 2



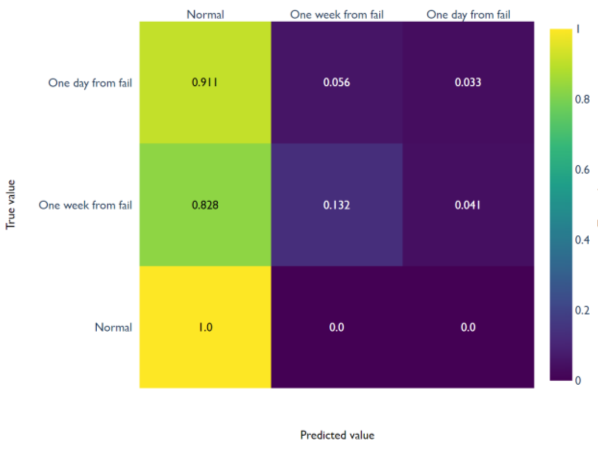


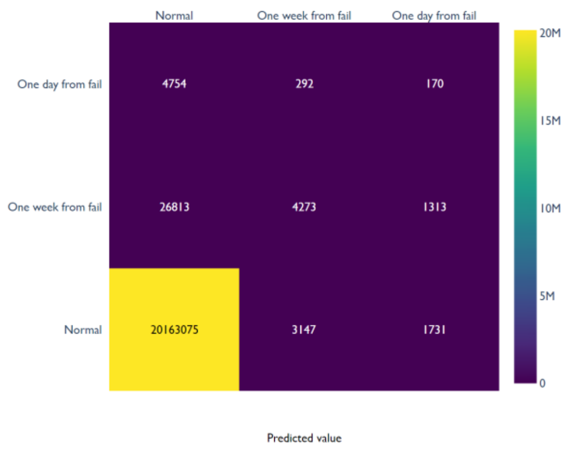
GRU model 3



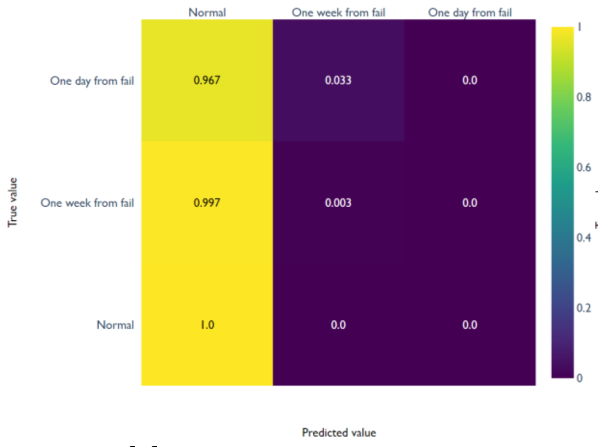


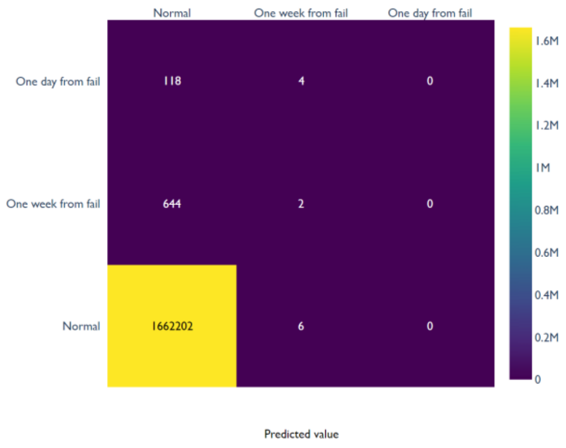
GRU model 4



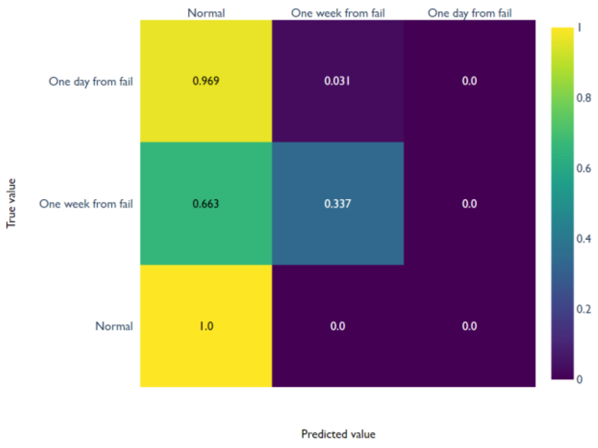


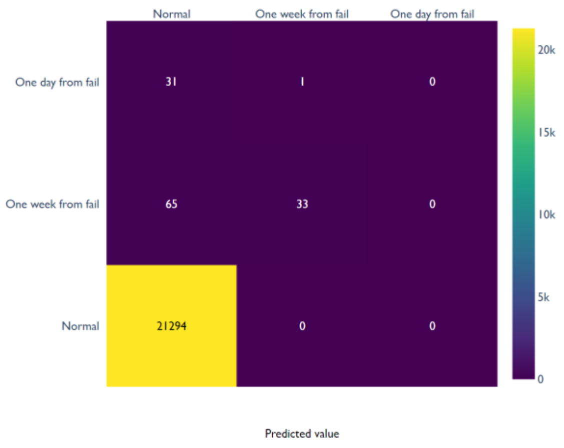
GRU model 5



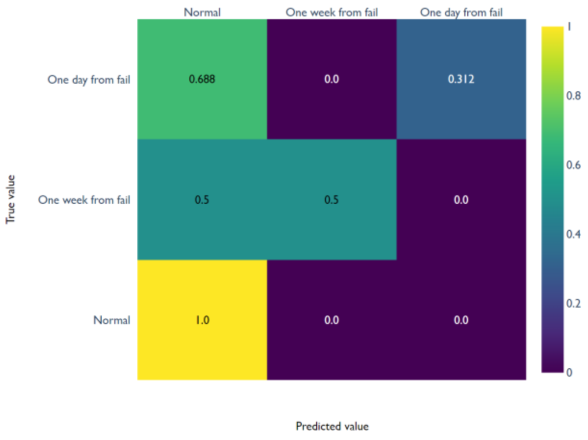


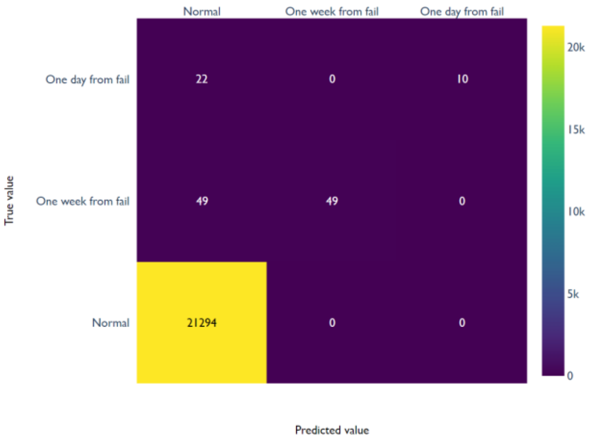
GRU model 6



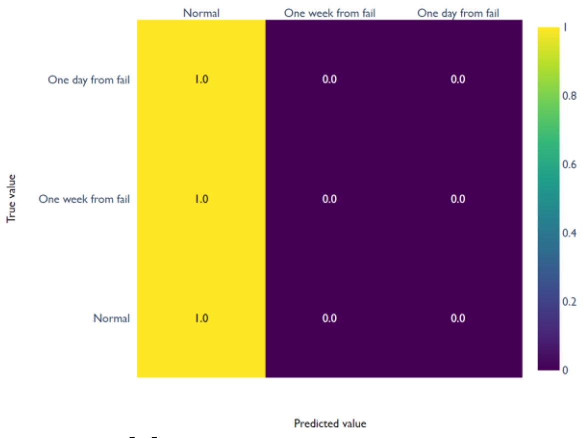


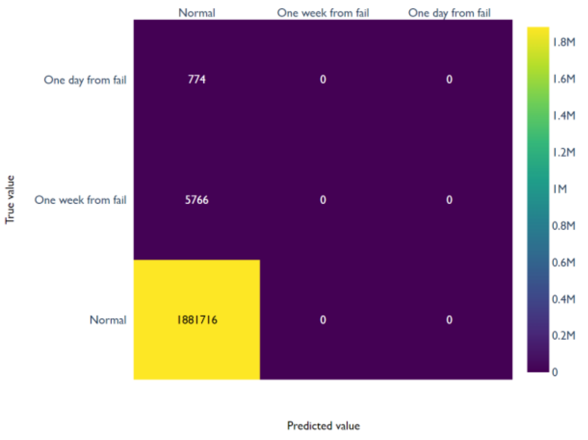
GRU model 7



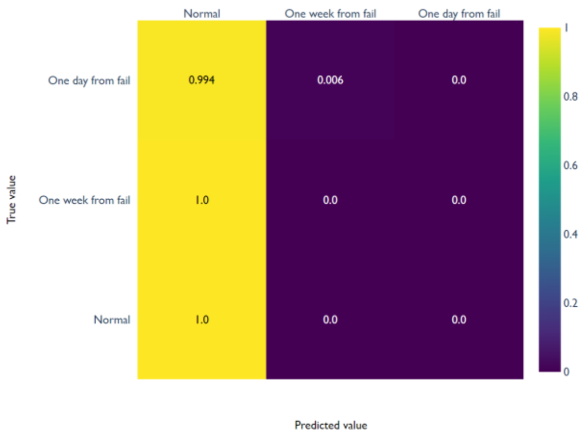


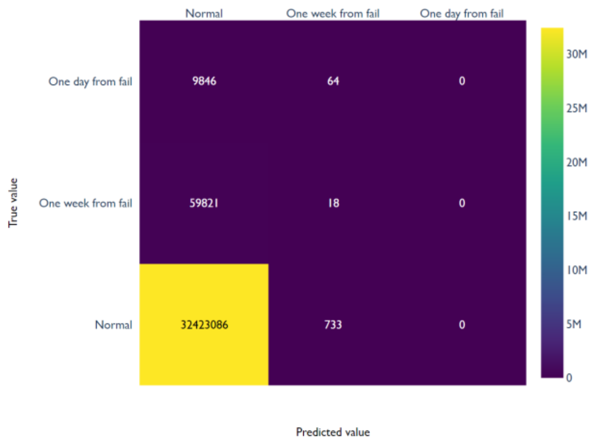
GRU model 8



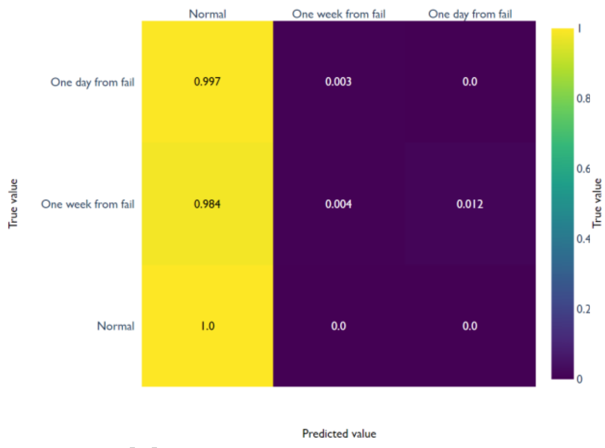


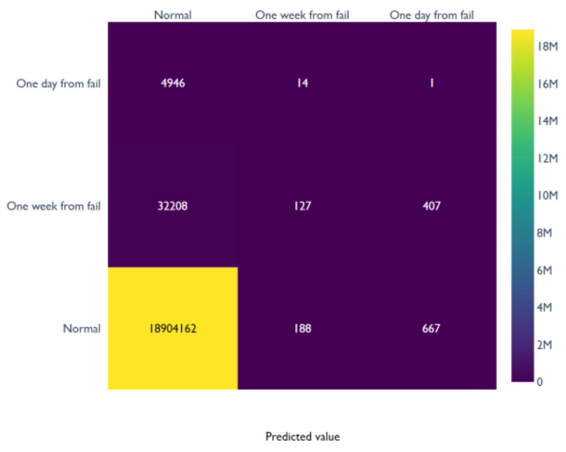
GRU model 9



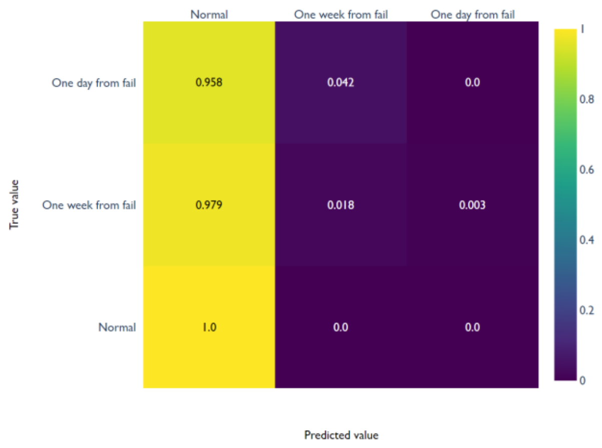


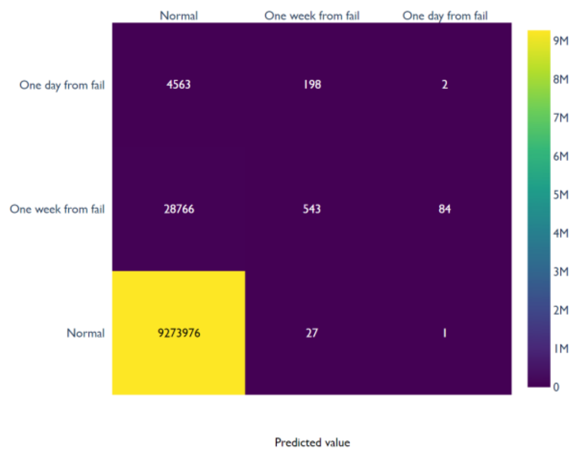
GRU model 10



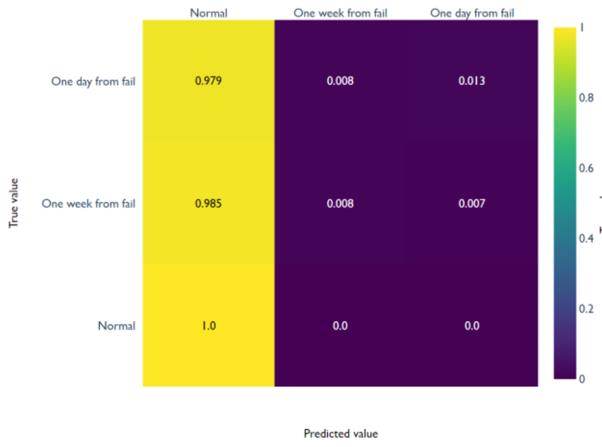


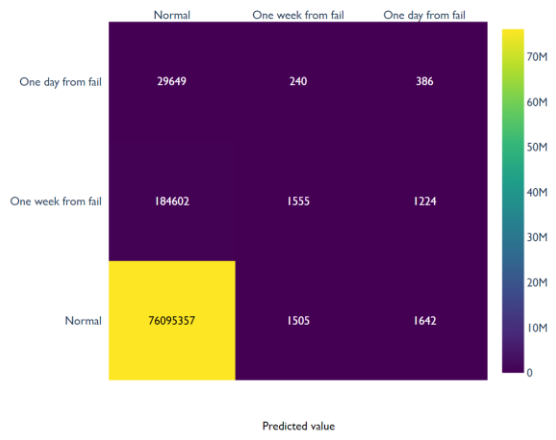
GRU model 11



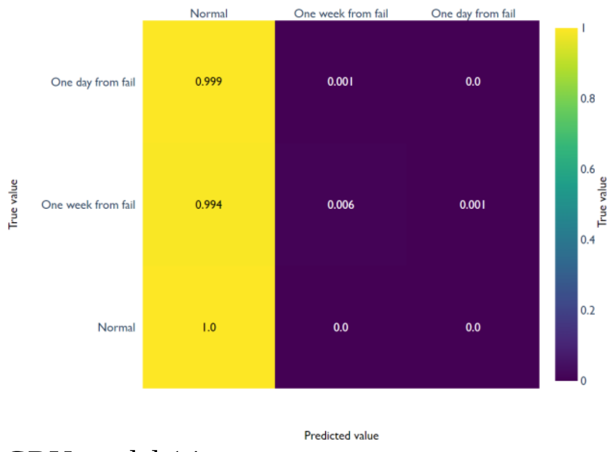


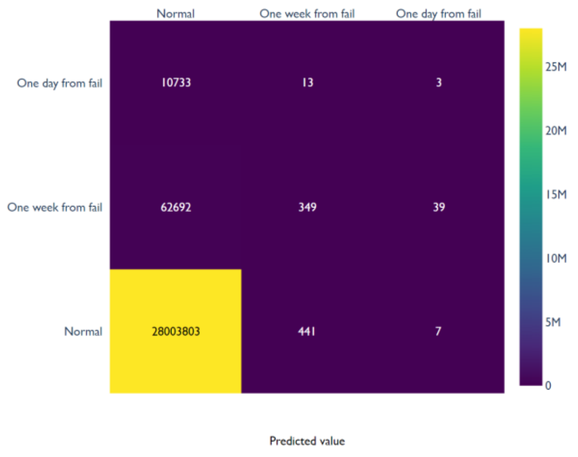
GRU model 12



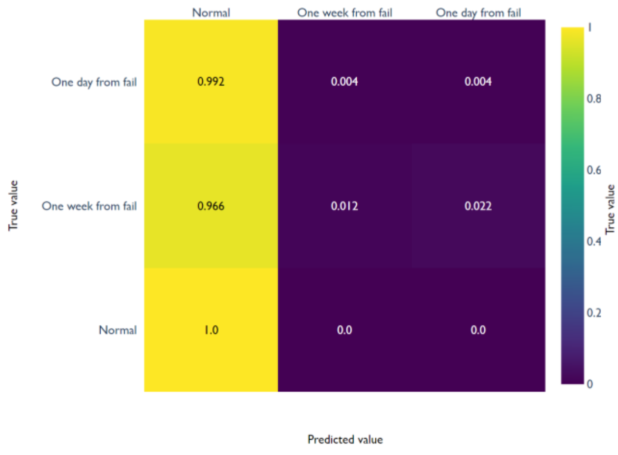


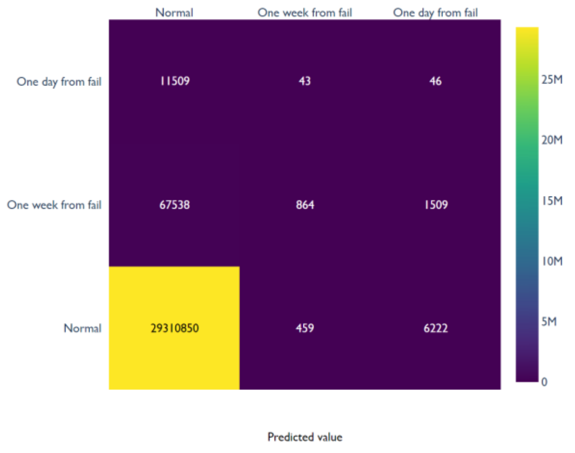
GRU model 13



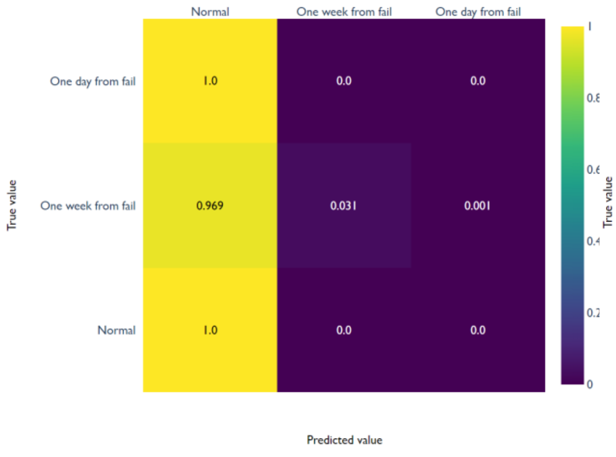


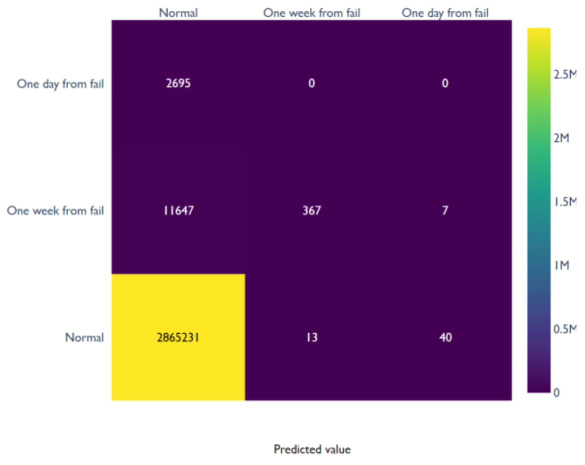
GRU model 14



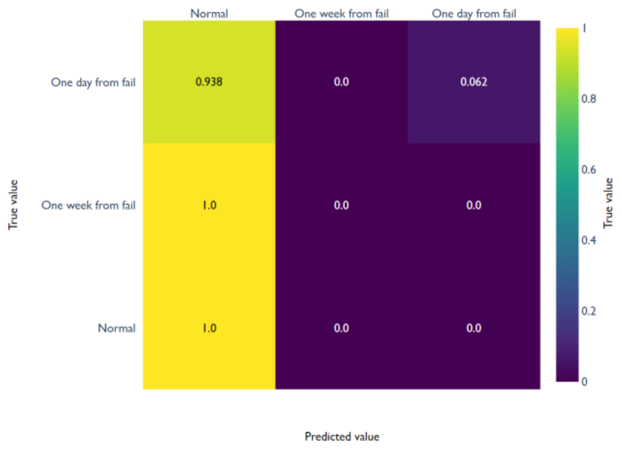


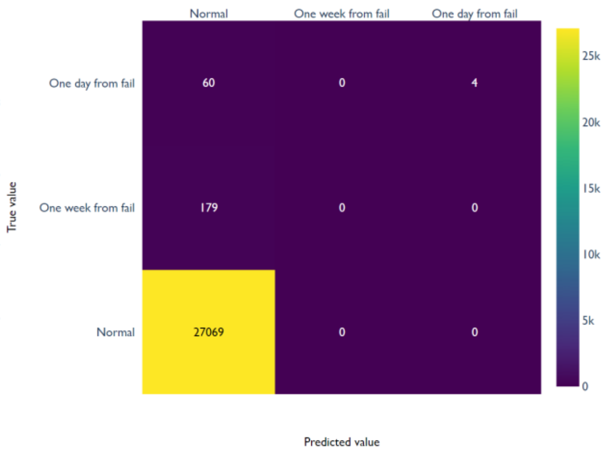
GRU model 15





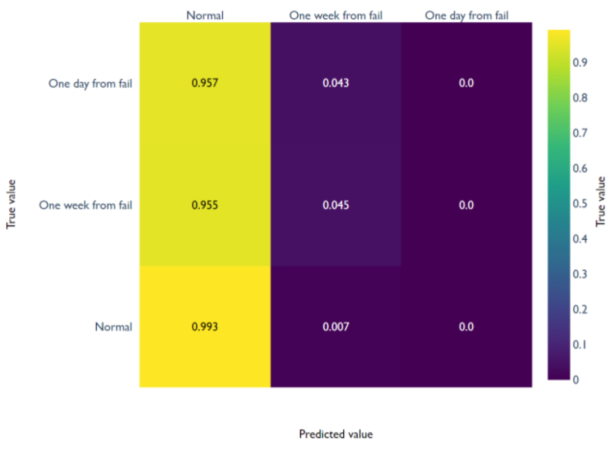
GRU model 16

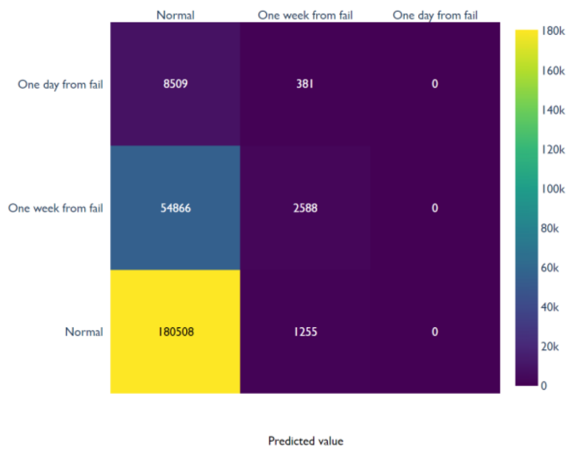

Normal
One week from fail
One day from fail
True value
Predicted value
0.938
0.0
0.062
1.0
0.0
0.0
1.0
0.0
0.0
1
0.8
0.6
0.4
0.2
0


Normal
One week from fail
One day from fail
True value
Predicted value
60
0
4
179
0
0
27069
0
0
25k
20k
15k
10k
5k
0

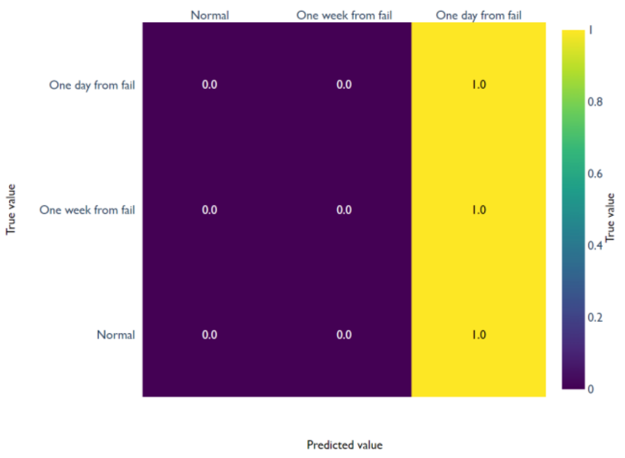
Appendix E. All support vector machine results in confusion matrices

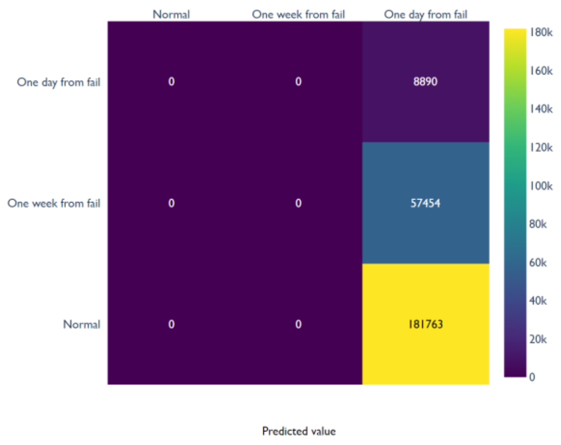
SVM model 1



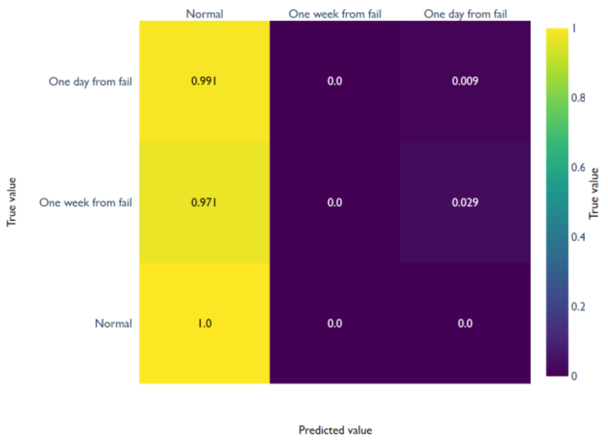


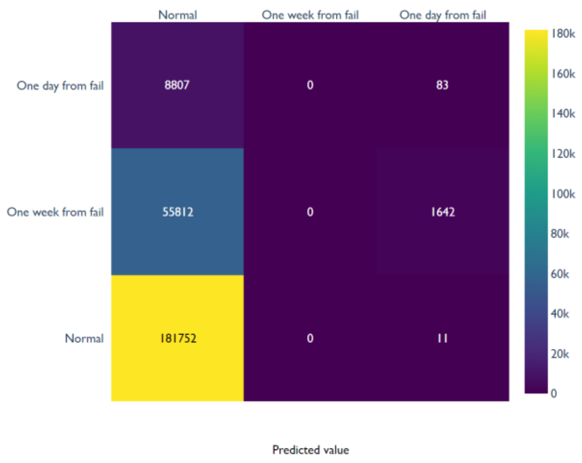
SVM model 2



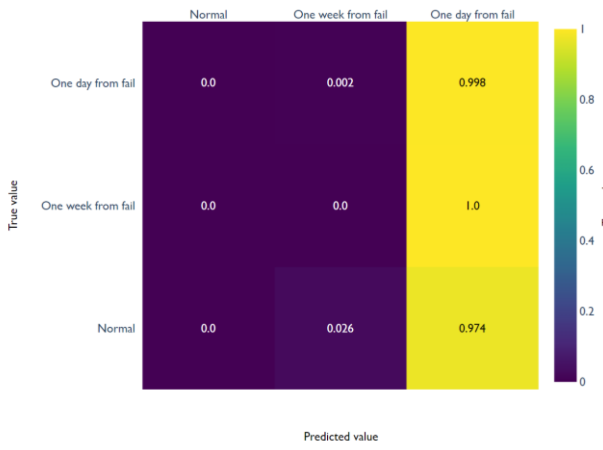


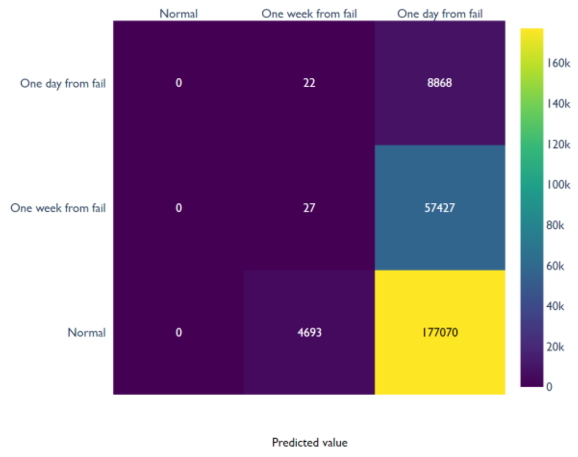
SVM model 3



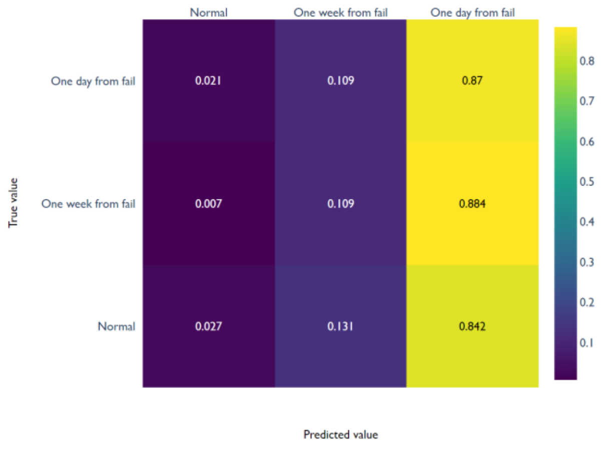


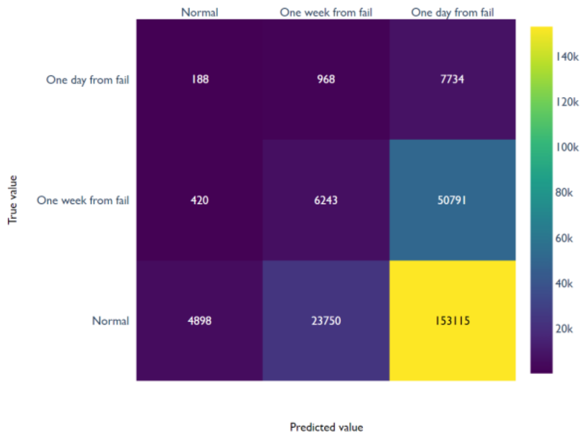
SVM model 6



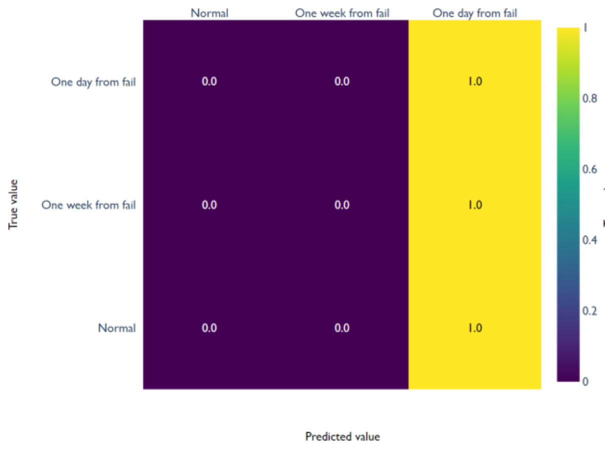


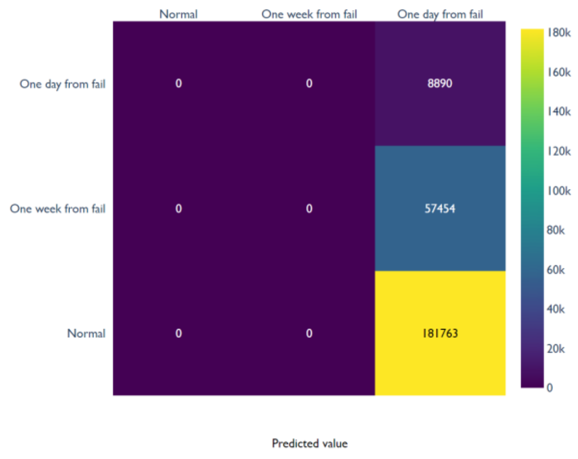
SVM model 7



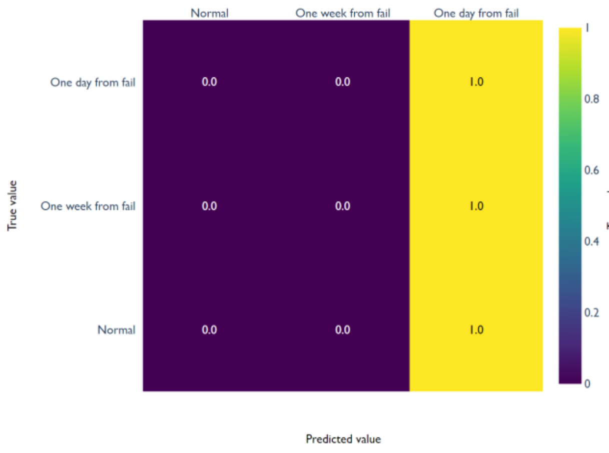


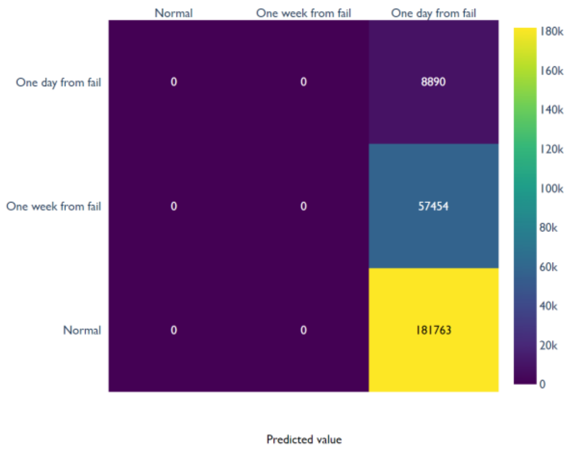
SVM model 10



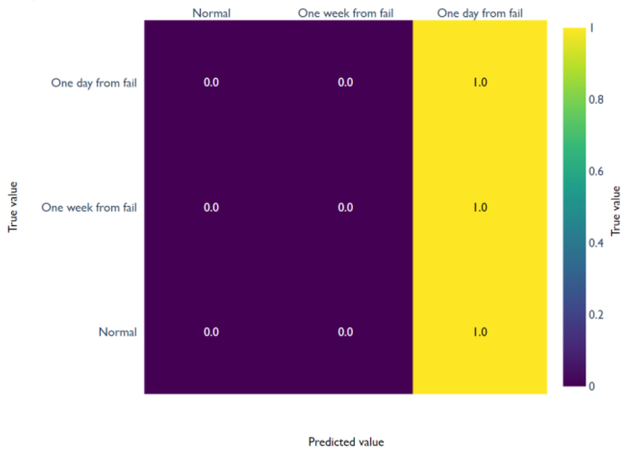


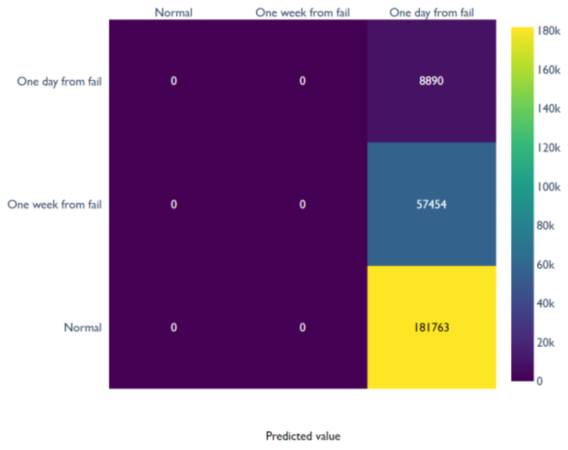
SVM model 11



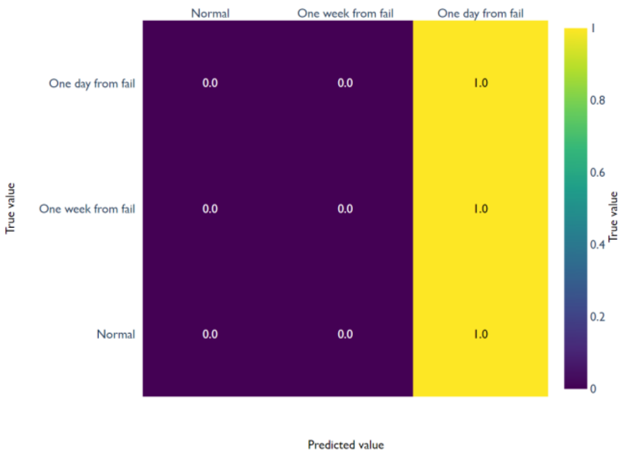


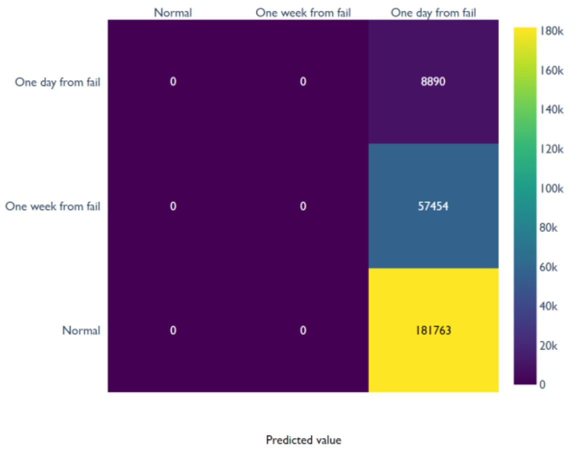
SVM model 12



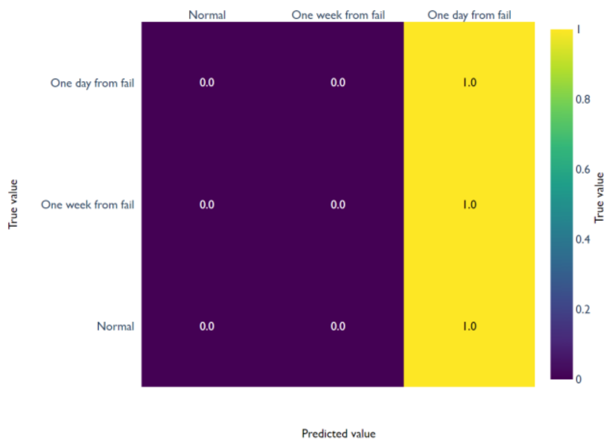


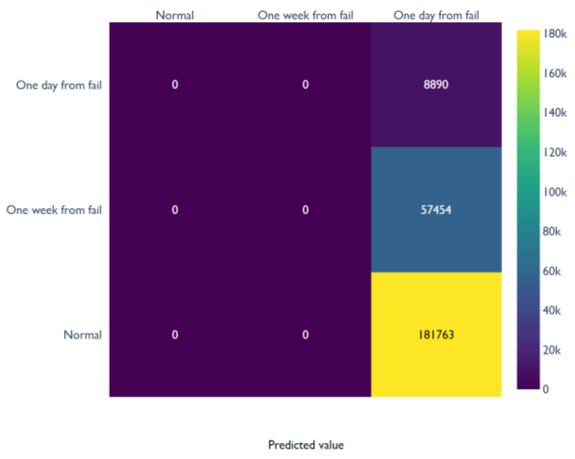
SVM model 13



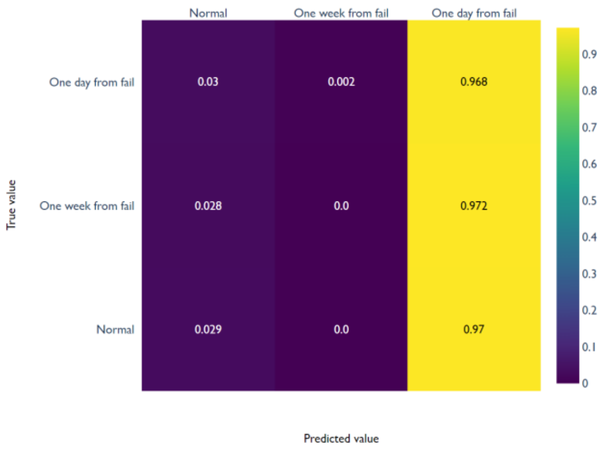


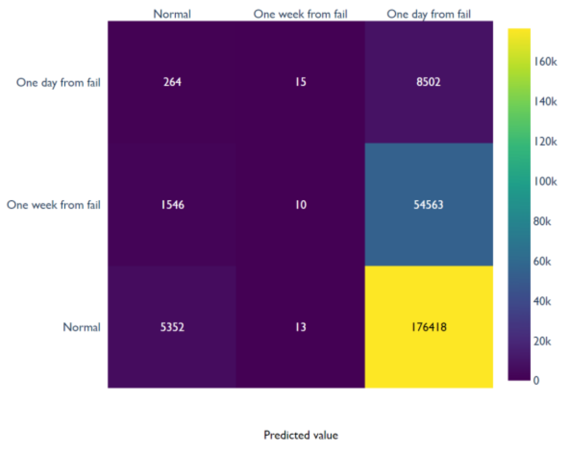
SVM model 14



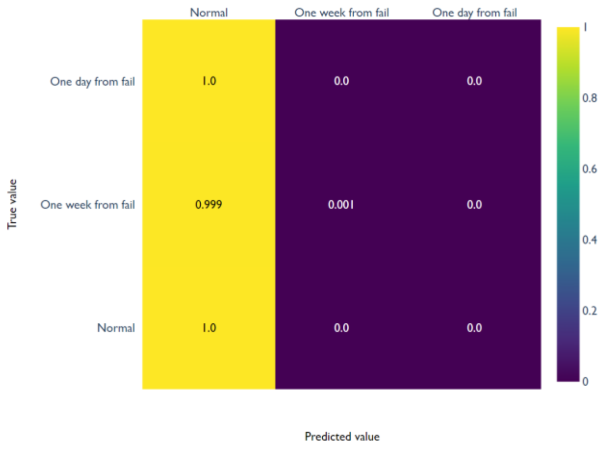


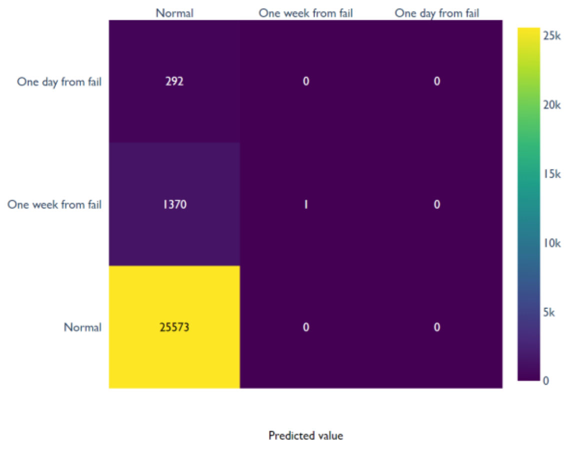
SVM model 15



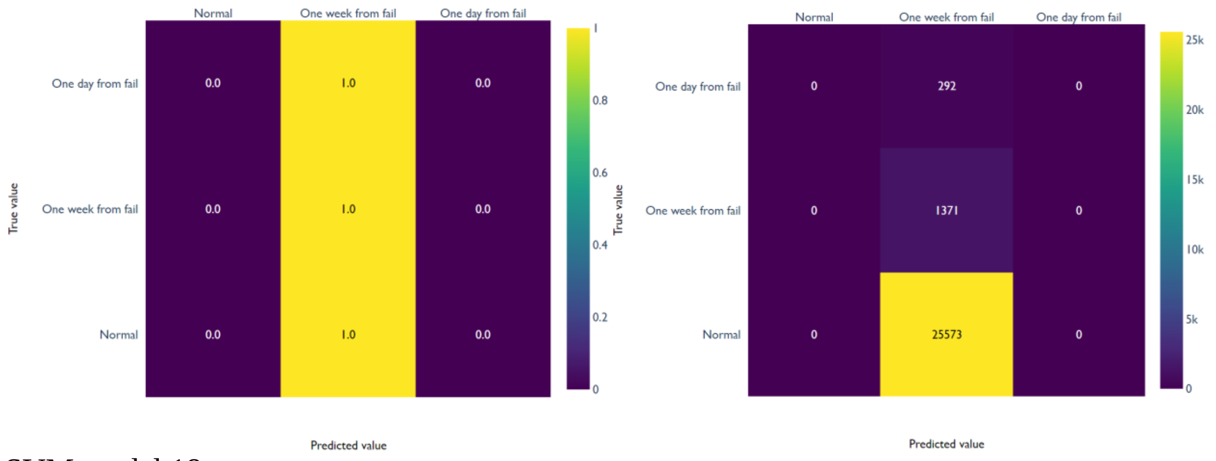


SVM model 16

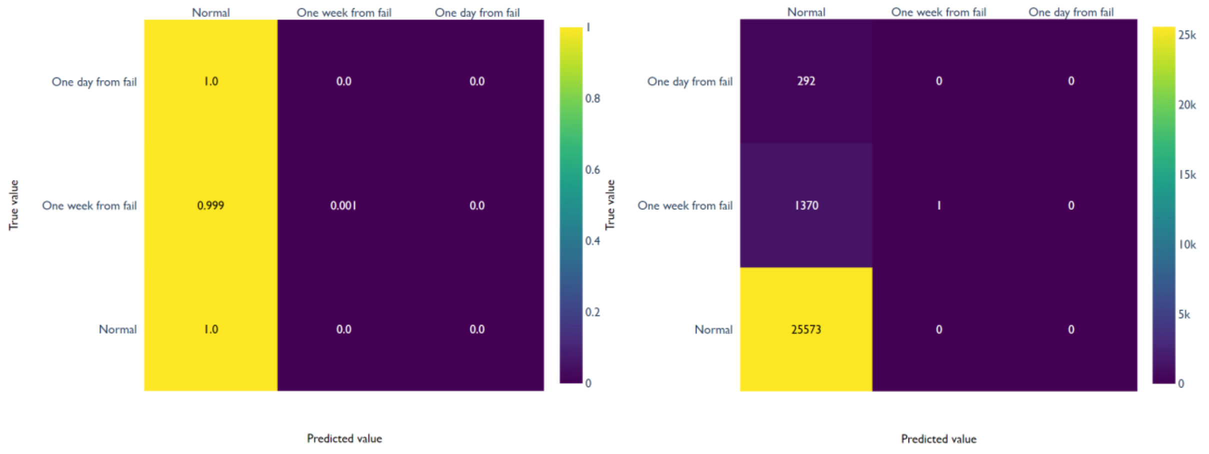




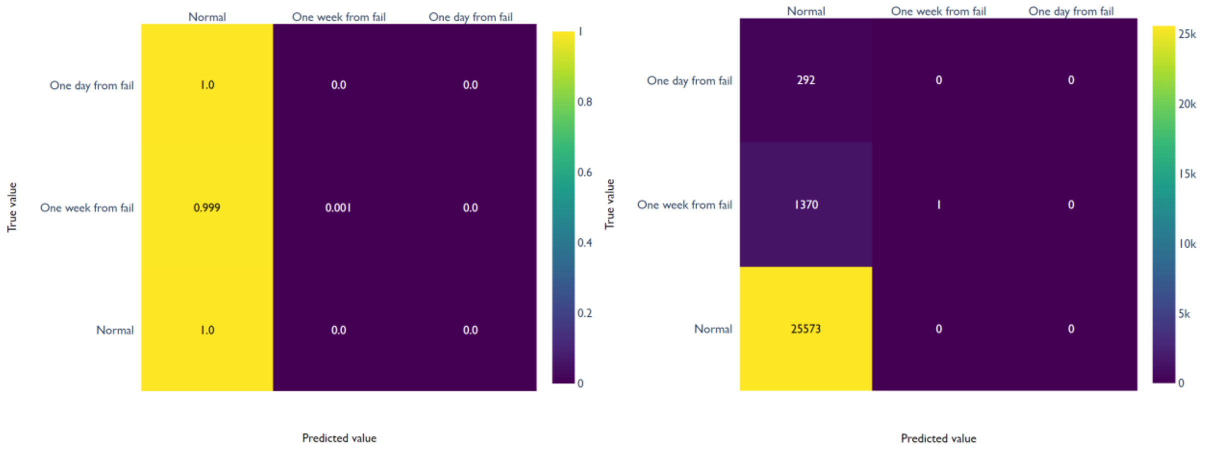
SVM model 17



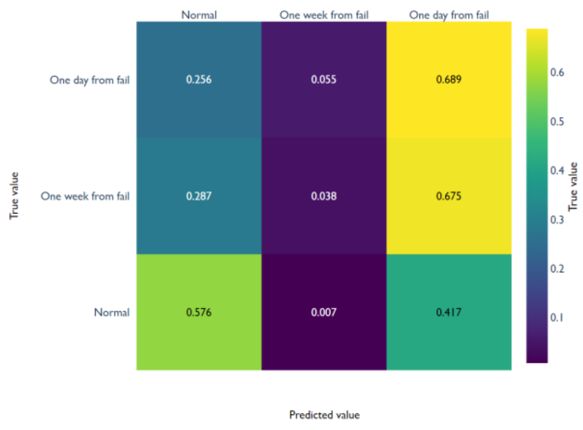
SVM model 18

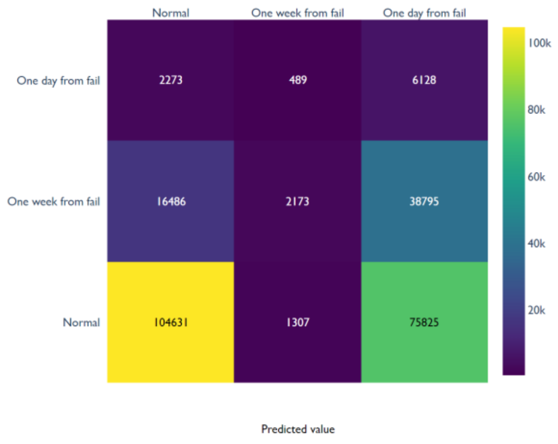


SVM model 19



SVM model 20


Normal
One week from fail
One day from fail
True value
One day from fail
One week from fail
Normal
0.256
0.055
0.689
0.287
0.038
0.675
0.576
0.007
0.417
0.6
0.5
0.4
0.3
0.2
0.1
Predicted value


Normal
One week from fail
One day from fail
True value
One day from fail
One week from fail
Normal
2273
489
6128
16486
2173
38795
104631
1307
75825
100k
80k
60k
40k
20k
Predicted value