# MASTER'S THESIS

## Data and System Health Monitoring in an Industrial Network

Charpentier, A.

**Award date:**
2022

# DATA AND SYSTEM HEALTH MONITORING IN AN INDUSTRIAL NETWORK

by

**Steven A. C. J. Charpentier**

in partial fulfillment of the requirements for the degree of

**Master of Science**
in Software Engineering

at the Open University, Faculty of Science
Master Software Engineering
to be defended publicly on Thursday May 19, 2022 at 09:00 PM.

| | | |
|---|---|---|
| Student number: | | |
| Course code: | IM9906 | |
| Thesis committee: | Dr. Stefano Bromuri (chairman), | Open University |
| | Dr.Ir Clara Maathuis (supervisor), | Open University |

# CONTENTS

# 1

# ACKNOWLEDGEMENTS

Whilst a thesis is primarily the culmination of 1 person's masters degree, it cannot be solely credited to any one individual. Without the hard work and support of a great many of people I would not be in a position to write these words.

Firstly I would like to thank my supervisors, Stefano and Clara. Your support, knowledge and patience were invaluable to me. Stefano, your frequent consultations helped me stay motivated and informed, and greatly contributed to the amount of knowledge I gathered over the course of researching and writing this thesis. I cannot imagine any better guides on this journey.

Secondly I would like to thank Bayer Antwerp, who, represented by Danny Van Looveren, allowed me access to their testing systems and wealth of information to use in the development and testing of my models. I would also like to thank my employer Magion for their continued support in my studies, and my colleagues for their insight and mental sparring keeping me sharp.

Thirdly, and perhaps most importantly, I would like to thank the people closest to me. My parents, who supported me through the time my educational journey took a substantial detour, in the interest of keeping my curiosity alive. Without this I would not be who I am today and might not have the drive to find what else can be learned. I would like to thank my in-laws for their interest, and my sister in law for her mathematical insight, helping me with tackle a few of the more abstract concepts. And perhaps most of all I would like to thank my girlfriend Dana, without her love, support and example I could not have successfully started or ended my masters studies.

# 2

# SUMMARY

## 2.1. DATA AND SYSTEM HEALTH MONITORING IN AN INDUSTRIAL NETWORK

To be able to manufacture goods, an industrial site depends on a high availability of its sensor data. When this data gets interrupted it might lead to problems when generating reports, or wrong actions being taken by operators relying on outdated data. The research in this thesis is aimed at maximising this availability.

Because earlier methods at rule based error detection failed or were to high maintenance we attempted to solve this detection issue by using machine learning techniques. In this research we compare 3 different machine learning techniques on data combined from historically captured quality data and captured network data.

We used the CRISP-DM method of data mining to find what data can be combined to form our model. The models we selected to work with are SVM, LSTM and Wavenet. These models were chosen because they each represent a different method of model organisation.

The model that had the most promise was selected to be integrated in a monitoring application. This monitoring application is a microservice based architecture with data collection services, a communication service, a configuration service and an analysis service. The model is loaded in the analysis service and can be updated or replaced at any time by the configuration service. The data collection services can be independently added to the application to extend monitoring capabilities and the communication service sends alerts when the analysis service detects an issue or when it fails to connect to one or more peers on the network.

## 2.2. DATA- EN SYSTEEMSTABILITEITS-BEWAKING OP EEN INDUSTRIEEL NETWERK

Om goederen te kunnen produceren is een industriele site afhankelijk van de beschikbaarheid van zijn sensor data. Als deze data onderbroken wordt kan dit leiden tot problemen met het genereren van rapporten, of verkeerde acties die genomen worden door operatoren die op verouderde informatie afgaan. Het research in deze thesis is gericht op het maximaliseren van deze beschikbaarheid.

Omdat eerdere methodes met rule-based fout-opsporing faalden of teveel werk met zich meebrachten, trachten we dit probleem op te lossen met behulp van machine learning technieken. In dit research vergelijken we 3 verschillende machine learning technieken om historische kwaliteitsdata te combineren met netwerk data.

We gebruiken voor de analyse van deze data de CRISP-DM methode. De modellen die we kozen om mee te werken zijn SVM, LSTM en Wavenet. We kozen deze modellen omdat ze elk een verschillende methode van model organisatie voorstellen.

Het model dat hier als meest veelbelovend uitkwam moet dan geintegreerd worden in een monitoring applicatie. Deze monitoring applicatie is gebaseerd op een microservice architectuur bestaande uit data collectie services, een communicatie service, een configuratie service en een analyse service. Het model wordt geladen in de analyse service en kan ten allen tijde geupdate of vervangen worden door de configuratie service. De data collectie services kunnen onafhankelijk toegevoegd worden aan de applicatie om de monitoring mogelijkheiden uit te breiden. De communicatie service stuurt alarmen uit als de analyse service een probleem ontdekt of als de service geen verbinding kan maken met één of meer zijn peers op het netwerk.

# 3

# INTRODUCTION

This research stems from the need to operate and maintain an industrial data management system in an environment where the loss of data can be critical but hard to notice. This data management system is centered around a data historian, which is a time series database used for retaining time sensitive data in an industrial environment. A data historian can have, often through a DCS, connections to a large number of distributed systems and they in turn can have multiple sensors that can be added or removed ad-hoc. Writing a rule based system to check for possible data loss would require a lot of time and effort to create and would have to be diligently maintained. At the site where we will conduct our study, multiple such attempts at rule-based data loss detection have already been attempted but have all failed due to the large upkeep of such a system. As the large number of false positives these system eventually created leads to the system being ignored.

This is where the use of machine learning or deep learning can be really valuable. Machine learning is using statistical algorithms to generate a model that can monitor and classify data based on this model [Samuel, 1959]. Deep learning is a subset of machine learning based on the concept of neural networks that continuously keeps learning and can adapt to changing environments [Schmidhuber, 2015]. If an algorithm can be trained on when to generate alerts it could remove the need to maintain an expensive rule based system that still requires a lot of manual checks even when performing optimally.
For this research we will be working with an industrial partner that has allowed access to their data and systems to build models and build a proof of concept, that might later be expanded to a fully operational suite. The goal of the research, as it stands, is to do a comparison on a few algorithms to find if it is feasible to create a model that can detect specific connection problems based on network traffic and already generated metadata on a select set of servers to peripheral devices.
In comparing the efficacy of a selection of popular machine learning techniques and using them to monitor stability in a network, we hope to increase the general knowledge on the use of machine learning, especially in a field where network instability might be hard to detect but have a large impact when ignored for too long and the use of machine learning on the metadata surrounding the actual data collected for process use.
The business partner in the case study is a data collection service that handles historical

data for the sensors and instrumentation for a number of distinct production units on an industrial site.

The intention is to test a number of algorithms with a wide spectrum of abilities including at least a type of neural net and a type of static algorithm. On a test system that was provided with real data (in this case a data historian server, data collector and simulated or duplicated data from DCS systems), but can be turned on and off to simulate faults for the system to detect.
This would serve as both a test for the algorithms and as a proof of concept for a monitoring application based on a microservice architecture that would continuously monitor live systems for unexpected data loss due to connections problems. These test scenarios would be created on a company owned environment and all data that leaves the company network will be scrubbed as to not violate company policy. The monitoring service architecture would consist of individual machine learning nodes on data collection hubs that would monitor data on their own system but also have a peer to peer (P2P) connection between all such services on the network (or a specific set via configuration) where each individual service communicates its status and the status of its monitored connection to its peers. This way the services also monitor each-other and can send an alarm on any irregularity.

## 3.1. STRUCTURE OF THE THESIS

First we will present a glossary with keywords that are important to, and will be used in, this thesis. In chapter 4 we will discuss The context for this research. Here we will give a general overview on machine learning algorithms and the relevant areas of network composition and hierarchy.
In chapter 5 we will discuss other work in relevant fields and where this thesis is situated in the collective body of existing work.
In chapter 6 we will discuss the data we base our research on. We discuss the sources and format of the data and the operations required to make the data usable in the execution of this thesis.
In chapter 7 we pose our research questions and explain the research model we will be using.
In chapter 8 we will discuss the evaluation and implementation of our machine learning models. We will discuss what criteria we will be using and which models we implemented for what reason.
In chapter 9 we posit an architecture that would fulfill the requirements we set to monitor network and system health and explain the reasoning behind the design choices we made.
In chapter 10 we discuss our conclusion and the possible future work we see following our research.

## 3.2. GLOSSARY

This section contains a general overview on the a number of concepts used in this document and what is meant by them in this context.

| | |
|---|---|
| **labelling** | This is the concept of adding meaningful metadata to a datapoint that is used to train an AI to correctly categorise said datapoint. |
| **microservice** | A service running on a system with loose coupling to other services with a distinct function that can be seen as part of an application. |
| **microservice architecture** | An architecture built on single responsibility services that makes a solution more easy to scale. Each individual service is part of the greater whole. |
| **IoT** | (Internet of Things) The concept of a network of physical object that each contain sensor data and software to communicate on the network. |
| **IIoT** | (Industrial Internet of Things) IoT as applied to industrial systems. |
| **OT** | (Operations Technology) All technology used in manufacturing systems. The difference between IT and OT is that OT often has more proprietary communication. and often functions as small islands to optimize system security. |
| **network traffic** | The traffic generated by the communication between two systems on the network. This can be monitored on either end of the communication pipeline and give an approximate idea of the amount of data that is exchanged between these systems. |
| **pattern drop** | A diminished change in the pattern of network traffic the AI system is monitoring. Eg.: if the normal network traffic on a port is between 10kb/s and 50kb/s, a pattern drop would be a period where the network traffic does not exceed 5kb/s for a longer than previously measured period. |

| | |
|---|---|
| **DCS** | (Distributed Control System) A control system for a production unit that manages the control flow for a production process. |
| **historian** | A database specialised in storing time-series data. The data stored by a historian is organised in such a way that data can be easily retrieved for analysis and cannot be altered once written. |
| **backfilling** | Adding buffered datapoints in the history of a historian database. |
| **ML** | (Machine Learning) The field of study focusing on data analytics by use of self and/or guided -learning computer algorithms. |
| **ANN** | (Artificial Neural Network) A category of AI algorithms based on the way biological neurons work. |
| **RF** | (Random Forest) A learning method for classifying data where decision trees a generated from training data and classification is done based on average predictions from the individuals trees. |
| **underfitting** | The concept of creating a machine learning algorithm not attuned enough to the data following in a large number of false positives. |
| **overfitting** | The concept of creating a machine learning algorithm too attuned to the training data causing a large number of false negatives. |
| **CRISP DM** | Cross-industry standard process for data mining. |
| **ODBC** | (Open Database Connectivity) A protocol created for and used to standardise communication with databases. |
| **REST** | (representational state transfer) A software architectural style that formalises sending or requesting data over the HTTP protocol. |

| | |
|---|---|
| **MAC address** | (Media Access Control) A unique identifier for any device connected to most types of network. |
| **IP address** | (Internet Protocol address) An address used to connect to a device on a network. |
| **ARP** | (Address Resolution Protocol) A protocol used by routers to link MAC addresses to IP adresses. |
| **TCP** | (Transort Control Protocol) A protocol used to send targetted messages over a network. |
| **ROC** | (receiver operating characteristic) A graphical plot that trends the ratio between true positives and false positives for a classification threshold value. |

# 4

# SCIENTIFIC CONTEXT

In this chapter we will discuss some background information out machine learning and machine learning techniques we applied in the execution of this research. We will also discuss other work that has been done on the subject of using machine learning algorithms to evaluate time series data in general and network or dataloss detection specifically. As the research itself leans towards the use of metadata we will address the importance of extensive and persistent keeping of metadata throughout this paper.
We will also focus on the use of a microservice architecture as opposed to a monolithic application to create and maintain a distributed monitoring system.

## 4.1. MACHINE LEARNING

### 4.1.1. MACHINE LEARNING PRINCIPLES AND SUBDIVISIONS

Machine learning is the abstraction of processing data by use of descriptive or predictive mathematical models where the processing algorithm is created by a machine based on information learned from an existing dataset.
Machine learning can be done in different ways and the problems or issues in the field can be subdivided in a couple of categories. Machine learning can be divided by the way the model is trained and maintained into supervised, unsupervised or semi supervised. Supervised algorithms are models based on a training set and clearly defined categories. A supervised algorithm easily translates into a static model. A static model does not change after training and does not have any flexibility after being created.

With a well defined training set and fully labelled categories these can be highly successful in a narrow application.
Unsupervised algorithms are more useful for data that lacks labels and structure. These can be used to discover methods to categorise data or to model data for which no categories can be defined. Unsupervised learning also allows for the use of dynamic models, where the model is continuously retrained on new data. The strength of these techniques

is that there is less effort or knowledge needed of the data under scrutiny and the model can detect possible unknown correlations or causation. It can also be widely applied due to the plasticity of the principles it is based upon. The weakness of these techniques are that a model created using unsupervised learning can be less accurate than a static model.

There is also a gray zone between these categories. This gray zone is semi-supervised, or weak supervision. These are terms used to describe machine learning methods where some labelling is present but is either intentionally or through circumstances incomplete. A semi-supervised technique uses a limited set of labels to optimally detect labels for the remainder of the training set. This can then be used to keep a continuously updated model running and steered in the right direction by manual input or to fill in the labels on a training set that can then be used as training data for another model.

Something our research might also touch on is transfer learning. This is the concept of using a model trained on a set of data as the basis for another similar set of data. For this research this would mostly apply to the difficulty of deploying a monitoring system on multiple nodes in a system that can be used to detect dataloss issues and cross-train detected signatures to other nodes. The more likely scenario is that this is a subject that would be very interesting in follow up research.

### 4.1.2. MACHINE LEARNING ALGORITHMS

Here we will expand upon a number of machine learning algorithms. All of these algorithms are widely used and researched and have their strenghts and weaknesses. We will only describe the more generally known examples of machine learning as the full list could fill libraries and most can be categorised as similar enough to one of the examples. Therefore we will describe these algorithms as archetypes where similar ones would follow the same of similar patterns.

### K-NN

kNN (k Nearest Neighour) classifies new datapoints based on their proximity to already existing datapoints. k is a positive integer that denotes the number of existing datapoints to consider. A k-value of 1 means the datapoint is classified as it's nearest neighbour, for a k-value of 5 the new datapoint is classified as the category with the majority of datapoints amonst the 5 nearest. This is a simple algorithm but does not scale well. The plots in figure 4.1 visualise the possible difference when using a k of 1 vs a k of 3 neighours.

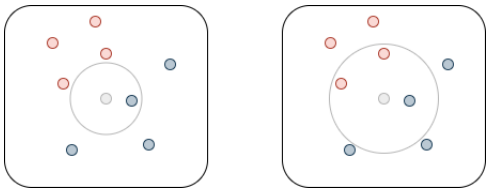

Figure 4.1: k-NN visualisation

### Naive Bayes

This classification method is based on the bayes probability theorem described in [Swarnkar and Hubballi, 2016]. A value can be predicted based on the likelihood of the value being present in conjuncture with other values for which a value is known. At first this might appear similar to the k-NN but the k-NN will compare a new datapoint with exisiting datapoints and find the closest. From this follows that k-NN does not scale well, as each new datapoints will have more other datapoints to compare with. The Naive Bayes method creates a function that outputs a probability. This will group a new datapoint together with the most likely probable match. This function may be retrained or updated, but will not require the calculation of every datapoint in the set for every new classification.

### Decision tree and random forest

A random forest is an algorithm consisting of a number of decision trees [Breiman, 2001]. When these trees are creating during the learning process they each have a different weight used to classify a datapoint [Breiman et al., 1984]. When a new datapoint is tested the random forest will output the result selected by most trees. A random forest has the ability to have multiple different classifications that can be set as the output of a datapoint. This makes it suitable for problems where more than two categories exist. A schematic example for a decision tree and random forest can be found in figure 4.2.

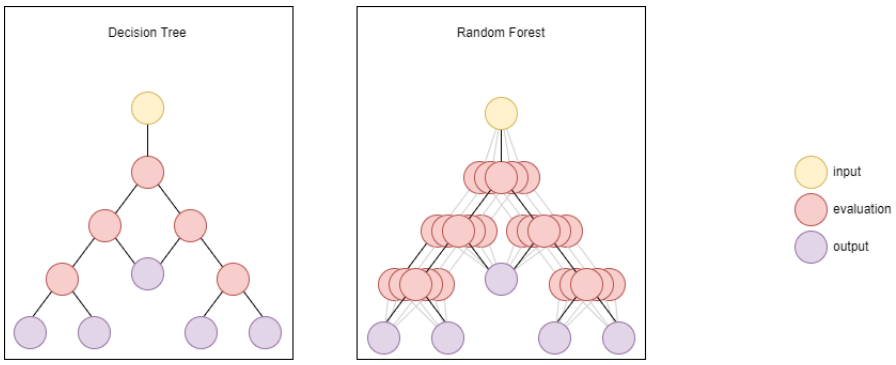


Figure 4.2: Decision tree and Random forest

### AUTOREGRESSION ALGORITHMS

An autoregression algorithm is a machine learning algorithm based on time series data where the next value in a series is predicted based on the previous values in the series combined with a random signal to predict a future result. Autoregression algorithms are used in statistical modelling and have their use mostly in field associated with nature, such as the forcasting of climatological data, and economics. For economics specifically Regis et al. [2022] collected a wide overview for autoregressive algorithms.

### SUPPORT VECTOR MACHINES

Support vector machines use a 'kernel trick' to draw a hyperplane that divides source data into two categories. It work well for data with a larger number of features and due to the kernel trick is relatively memory efficient [Razzak et al., 2020]. Support vector machines will be discussed in more detail in the model implementation section of this thesis.

### MULTILAYER PERCEPTRON

A multilayer perceptron can be seen as the most basic version of an Artificial Neural Network. It works by emulating the operation of a biological nervous system. A multilayer perceptron is composed from multiple cells arranged in layers where each cell has a weight and bias that determine how it will classify input data. When multiple layers of these cells are combined this network of neurons can be trained by altering the weight given to the individual cells based on their accuracy. Inaccurate cells will have their weight reduced and their output will become of lesser influence when the system as a whole generates an output. The result of this process is a system that can reliably categorise data but maintain some randomness to prevent overfitting.

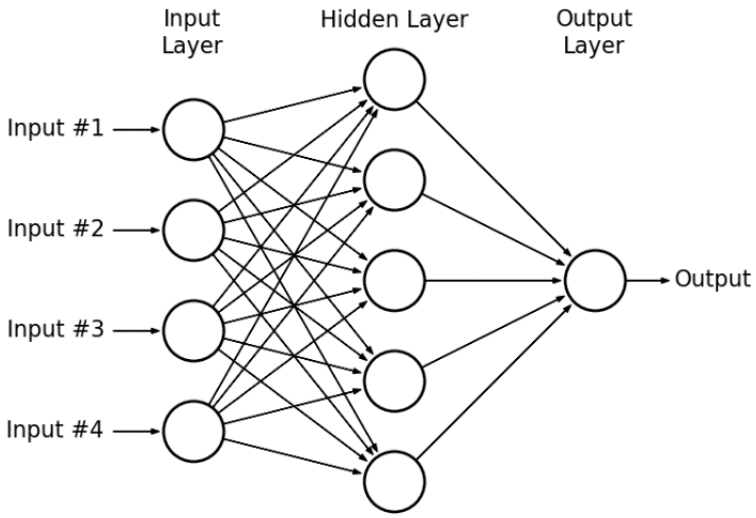


Figure 4.3: Representation of a multilayer perceptron as found in [Hassan et al., 2015]

### RNN (Recurrent Neural Network)

The RNN is a category of neural network where the input is sequential and where the nodes in the network have an internal state where the evaluation of each node is dependent on the previous state the node had. This increases the prediction power of the RNN in the case of time series data at the cost of introducing an internal state that can diverge into infinity [Bengio et al., 1994].

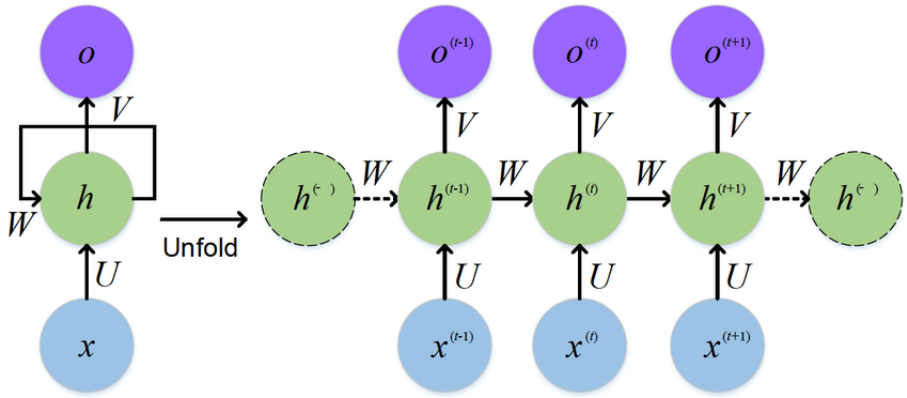


Figure 4.4: Representation of a multilayer perceptron as found in [Feng et al., 2017]

## 4.2. NETWORK DATA TRANSMISSION

The basic metric we will be using for our research is the transmission of data over a network. The manner in which computers send and receive data is crucial to understand how a monitoring system can be set up. Network architecture can be broken down in layers according to the OSI model in figure 4.5.


Figure 4.5: Open Systems Interconnection Model

The lowers layer defines the hardware and physical properties of the network. For the purpose of our research it is mostly important to know it exists. Every higher layer requires the lower layers to be present to function.
Each layer in this model increases the level of abstraction from an electronic signal up to a human readable interface. The second layer provides a protocol for the most basic level of communication over the hardware. This is where the linking is situated that allows networks to route data to and from the correct systems.
The next level up is the actual networking level. This is the level where addressed packets are created and distributed over the network. This level is not concerned with the actual routing (lower) or content of the message (higher). Its function is limited to the distribution of an addressed packet to a destination.
Supported by this layer is the next layer up. The transport layer is where connections between client and server are negotiated and the form of the message is defined.
The fifth layer defines the individual session. This includes the opening and closing of connections and defines the ports used for specific channels within an application.
The Presentation layer parses the data received from the network into streams and files the client can use for further processing. The last layer, also known as the application layer or end-user layer defines requests and replies that can be used by applications to provide end

users with meaningful data.
For this research we are mostly interested in layers 3 to 5. We will be expanding on these topics in the following subsections.

### 4.2.1. SECURITY

The existing framework surrounding the transmission of data over a network as discussed in the previous section provides a number of different methods of measuring data transfer over a network.
The research by [Diyeb et al., 2018] compares methods of hacking that can be used to gain information on the traffic and data on a network. There are measures that can be taken to scan the flow of data on an entire network. The process of network sniffing is a technique that intercepts packets moving on a network. On older networking environments, where devices are patched together with hubs this can work by accepting all packets on the network, even those not addressed to the device the sniffer is hosted on. This works because a network hub is not a routing component, meaning it does not send data to one specific IP in its address list, but it broadcasts all packets it receives to all connected devices. This works because the IP protocol has no built in security or reception acknowledgement. The targeted device on the hub will receive and acknowledge the connection and all other devices without a network sniffer will simply ignore the packets not addressed to them.
The hub network is for this reason an incredibly insecure network. Most modern systems work with routers and switches. These systems hold address tables they use to send data to a specific device. This means a network sniffer on a third device cannot access packets sent between 2 devices on a network without redirecting the data from the router or switch itself.
Redirecting this data is not impossible, as techniques like ARP poisoning exist. ARP poisoning works by abusing the ARP (Address Resolution Protocol), a protocol used by routers to link MAC addresses to IP, to redirect traffic to the sniffing device. This attack can also be defended against by implementing a static IP for all devices attempting to connect to the system. Using a static IP layout sacrifices flexibility for security and is something an industrial environment will likely implement.
The network traffic for a single device can be monitored using system handles and tools provided by the operating system. For windows this is WMI or Windows Management Interface [whims]. These interfaces use the same handles that are needed for the operating system to handle network sessions and are therefor always present. On the local system the only limitation on monitoring applications is user privileges.

### 4.2.2. Packet Allocation

As discussed earlier a network connection by the OSI model does not work in a single continuous stream of digital information. All data on a network is sent as a sequence of packets [Braden, 1989]. These packets have meta information to denote source and destination and for some protocols like TCP have a self-checking function to preserve integrity [tcp, 1974]. This meta information also includes destination ports that designate the packet as belonging to a specific application. This meta information can be used to track packets entering a device. The packets arriving on a server can be filtered by port and it's size can be measured.

## 4.3. Summary

As we understand it there is a lot of research surrounding the topics we chose to focus on. Machine learning can be done using a variety of techniques and models and these machine learning models can be integrated in a monitoring application using a number of architectures. The detection of network activity can be narrowed down to a specific stream to isolate away noise on the network and can be done in such a way as not to compromise the security of the system under observation.

# 5

# RELATED WORK

In this chapter we will discuss the related work that touches on the various topics discussed in this research. Most notably we will discuss the subject of monitoring dataflows on a network, the use of machine learning to interpret, predict and classify time series data and the use of a microservice architecture in an industrial setting.

## 5.1. MONITORING

Monitoring the stability of the dataflow in an industrial network can be done by different approaches. For this research we have chosen to analyse the dataflow between the process control layer and the business layer. In [Raposo et al., 2018] the researchers created services that run on IoT devices that self-monitor and send data to a centralised service where logs are kept for later analysis.

For more rigid process control networks that run for years without any modification [Valdes and Cheung, 2009] suggest an easier rule based approach to whitelist normal behaviour and detect issues. This method counts packets and packet sizes to determine normal data flow. Based on these variables they define rules that categorises dataflows into normal, anomalous or attack attempts.

## 5.2. MACHINE LEARNING FOR TIME SERIES DATA

The field of machine learning has explored a wide variety of possible algorithms to allow a computer to detect patterns [Nguyen and Armitage, 2008]. These algorithms are mature enough that they can be found implemented in multiple libraries and programming languages and be used by engineers without in depth knowledge about the mathematics involved. Examples of these libraries as they exist for the programming language Python can be found in Keras and Pytorch. Various papers exist that compare different machine learning methods as they are used in different fields and applications[Cui et al., 2018]. This

shows that, although machine learning has existed for a while and has a lot of applications, the specific strengths and weaknesses of each method can still be tested on different datasets to further broaden our collective knowledge on the subject.
A field where machine learning has made a large difference in interpretting time series data is text generation and sound analysis. Both of these applications have uses for the greater public and have helped in the development of machine learning algorithms that can be applied to other fields. One example of an algorithm that was designed for sound analysis is wavenet [van den Oord et al., 2016]. This algorithm was after its introduction used by google to create the voice of its digital assistant.

Machine learning algorithms have already been used and analysed in IoT and IIoT for detecting device types on the network. This to detect and classify new IoT devices connected to the network, [Meidan et al., 2017a] or detect and block unknown devices [Meidan et al., 2017b]. Machine learning has also been used to find malicious access on a network by analysing network traffic [Hasan et al., 2019] and to determine and detect the pattern of specific internet protocols such as P2P (peer to peer) communication [Murat Soysal, 2010]. Using machine learning to detect problems on an industrial network has also been proposed by Mantere et al. [2012]. This paper discusses advantages and disadvantages of using machine learning in industry as industrial networks can often be seen as closed systems where internet accesq is either prohibited or restricted. It also mentions noise created by maintenance on the network itself or adding and removing equipment as needed by the operation of the factory itself.

Machine learning can also be used in predictive maintenance, where the trained algorithm can alert to signals that a breakdown is imminent. To the best of our knowledge there is no related work that uses this on the stability of the data gathering tool itself, which this research will attempt. There is however information on how this has been achieved on sensor data in industry to predict failing equipment [Namuduri et al., 2020]. There are also algorithms specifically designed to work on time series data that indicate the feasibility of using machine learning to detect signatures in time series data as proposed by Munir et al. [2019] with DeepAnt and related Recurrent neural net (RNN) algorithms such as long short term memory (LSTM) as proposed by Hochreiter and Schmidhuber [1997] that can mitigate the effect of the vanishing gradient problem as Hochreiter describes it.

The idea of the importance of data loss prevention is not new [Liu and Kuhn, 2010] and has been proven to potentially lead to costs for the data owner. Data can be protected by various means, as explained in Liu and Kuhn [2010] and for this project we chose to specify monitoring network data as the focal point. A similar project has been done by Shipmon et al. [2017] where Google network traffic was analysed to detect possible data loss, here in a highly noisy sequence of network traffic datapoints. The research in Shipmon et al. [2017] was limited by the lack of labelling data we will attempt to overcome by using data historian bad data markers.

## 5.3. Machine learning in industrial settings

This research is not the first to use machine learning methods on network traffic, a survey of techniques used for this was compiled by Pacheco et al. [2019]. Most research however places its focus more on intrusion detection over other areas such as faulty equipment, human error or network fluctuations. When network traffic is monitored it is also often on the level of snooping the network itself. In this research we use a more limited data gathering sample where only the data to-and-from specific applications is monitored to keep network intrusion to a minimum.

## 5.4. Microservice architecture

The increased reliance on distributed microservices has moved some to consider methods of health monitoring for these services. An example of this is Jiang et al. [2020] where a system is suggested where microservices register with a monitoring system that can warn in the event of one of these microservices becoming unstable or generates an alarm that must be pushed. This however does not eliminate the possibility of a single point of failure disabling the system.

The microservice architecture is an extension of the service-oriented architecture. Which in itself is already often used in industry settings as discussed in the survey [Xu et al., 2014].

Using microservices has very specific advantages over the standard service-oriented architecture in that it allows for the use of multiple programming languages and technologies in a single application domain. [Krylovskiy et al., 2015] uses this to their advantage in creating an framework that allows the components of a smart city to communicate. This paper also discusses the specific pros and cons that can be found when using a microservice architecture.

The strength of microservices not locking a developer into a specific programming language or other tool is also discussed by [Thönes, 2015]. Where they also state the advantage of scalability and maintainability a microservice architecture has as compared to monolithic applications. They also discuss the legacy of monolithic applications in existing operations and the cost of maintaining these systems.
In figure 5.1 we compare the setup for a traditional application versus a microservice architecture. The microservice architecture is more involved and takes more time to set up compared to the monolith, but by having the responsibilities of each service closely delineated, the microservice architecture allows for the addition of additional business logic or an additional external interface withouth changing the existing, working, application. Each external access component, like a database, having an interfacing service, also allows the application to be independant of a specific source. If a database vendor stops the support for a legacy product, the decoupled nature of a microservice makes it possible to write an interface for a different database and substitute this service for the legacy connection without changing any part of the business logic of the remaining services. the

survey [Laigner et al., 2021] notes that databases are often abstracted in microservices, but developers lacking complete business understanding of the application being developed often create shortcuts when it surfaces that some data has to be combined from multiple databases. From this follows that these services, which should be loosely coupled, generate problems when refactoring because of these shortcuts. A refactoring in such a situation would require a larger analysis to attempt to decouple these services that might include changes to other services. This shows a possible pitfall when designing an application with microservices to have a full analysis of the dataflow within an application before starting the architecture design.

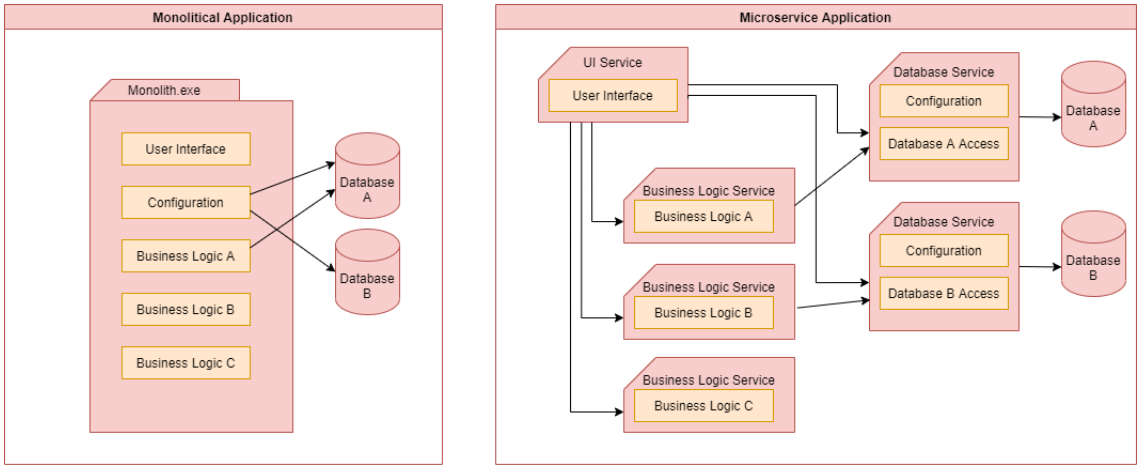


Figure 5.1: Monolithic application versus microservice architecture

## 5.5. SUMMARY

There is already a large amount of research into the operation of different types of machine learning models on network data and time series data. From what we found however the machine learning research into network data focuses more on data as a whole, meaning the detection of abnormal patterns that would indicate intrusion or tampering. From what we found nothing researches the data for a specific system or process. This granularity can be both a strength and weakness in our research. The isolation of external noise allows us to detect problems within the process stream more clearly, but might cause us to miss external factors.
The use of microservices also has some research indicating its uses and advantages in the environment we intend to use it. The maintenance and extensibility of a monitoring application is important in this setting because of the heterogenous nature of the environment where some instances might have different monitoring needs. On the other hand it is important for all services to be able to communicate their status with each other to create a robust system. This plasticity in function combined with a rigidity in communication matches with the specifications of a microservice architecture.

# 6

# RESEARCH METHOD

## 6.1. RESEARCH QUESTIONS

In order to achieve the stated research goal we formulate the following question:

**Can an AI microservice infrastructure be created to monitor network data and compile this data into a message to peers on a network to generate alerts when a pattern drop is detected?**

To answer this question we propose a set of subquestions:

- Rq1: How can recorded data from a data historian be linked to significant differences in network usage?

  This research question will be answered by finding statistically relevant connections between gaps in data historian tags, network traffic on a data collector service and/or reported system outages in operator feedback forms.

- Rq2: What algorithms are best to teach machine learning to monitor categorised data usage?

  There are a lot of different machine learning algorithms to consider when modelling a system. From a highly varied subset of these algorithms, namely LSTM, SVM and wavenet, we will test which would be the most accurate and most performant choice.

- Rq3: What is the best architecture to create an alert system to warn of a pattern drop and integrate a ML subroutine?

  A number of software design patterns exist that could potentially be used to create a functional, maintainable system as seen in course IM0203 Software architecture. The problem discussed here is best solved by using a

subset of these patterns. Weighing the advantages and disadvantages of these patterns can provide a clear design to structure the solution with.

- Rq4: What are the requirements to maintain an AI model in a microservice structure?

  An AI model needs to be maintained so it does not lose accuracy. We will try to find the best way to go about doing this in a disconnected overarching architecture.

## 6.2. Research model

The data collection section of the research will be based on the Cross-industry standard process for data mining (CRISP DM) method [Wirth, 2000], where business and data understanding drive the formation of the data models that will be used. This method was chosen as it is the industry standard for data processing and provides an open framework to work with.
This research will consist of the creation and labelling of data on a simulated system. The model derived from this will then be used to test a large amount of historical data from live systems. Once this model, through the CRISP-DM method, has been iterated to a point where its fidelity is high enough to warrant its use, it will be integrated into a service that will monitor data live and has the ability to transmit its status to its peers.

### 6.2.1. CRISP DM

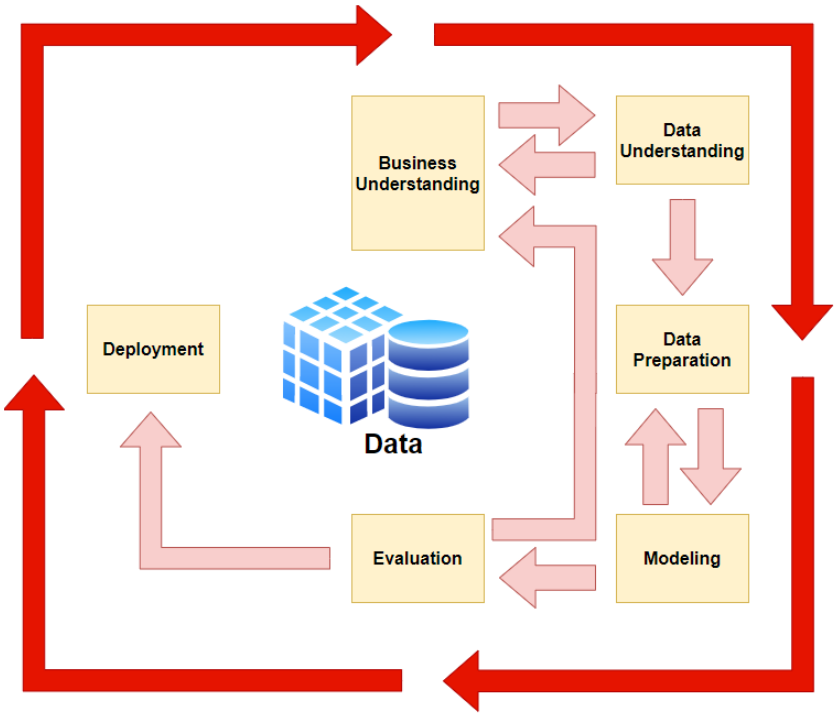


Figure 6.1: CRISP DM big picture

**Business understanding** Define what it means for a system to be malfunctioning or under performing and what exactly can be achieved within the scope of the project. As data is a critical component in the manufacturing facility where this project is undertaken, the undetected loss of a couple of sensors can cause a production process to be slowed or even halted. From experience it was often the case that before a more general system failure, data loss could already be found on a selection, but never or rarely the same selection, of datapoints.

**Data understanding** Map what data is needed and identify possible issues with this data. Find the location of the required data sources and test this data for quality and relevance to the business problem. For this project we will be focus on network traffic linked to gaps and upstarts in the historical data.

**Data Preparation** Clean the data that was collected by removing possible artefacts or non-representative data such as shutdown periods. Then collate this data in a way where it becomes useful for an AI algorithm. Here we will attempt to use network traffic data and historical data, combined with simulated fault states to generate a learning environment for the AI algorithms.

**Modeling** Select and train a selection of AI algorithms to find the best match for detection and diagnoses.

**Evaluation** The project will be evaluated on the success rate of the algorithms when ran against a set of evaluation data. If this rate is high enough to project will be further rolled out to a full alert suite, otherwise it can serve as a valuable lessons learned on the use of AI in this field.

**Deployment** This will only be partially implemented in the scope of the graduation project. A full deployment of the detection and cross-communication system would take a long time and would have rapidly diminished returns.

## 6.3. SUMMARY

To fulfill our research we will start from an understanding of our data. We will collect and select the data based on key meta information suspected to be interesting for the research and classify the data accordingly. Once we have run the data through some models we will evaluate the efficacy of the models in question. These models will then be judged on the effort required to train and a ratio between false positives and false negatives to determine the model most fit for purpose.

# 7

# EXPLORATORY DATA ANALYSIS AND MODEL IMPLEMENTATION

## 7.1. EXPLORATORY DATA ANALYSIS

To start the exploratory data analysis we attempted to identify visual patterns in existing trending data. When assembling the necessary data we searched through the databases using inbuilt tooling to search for problem areas and correlating trends. The data we were interested in mostly focussed on tag quality data, memory consumption, and system uptime. The nature of the historian is such that it backfilled most issues. The first attempt we made to look for areas of increased data loss was by searching for occurrences with an increased number of bad tag values.
These searches did indicate times of major physical changes, but were unable to locate failures, hypothetically this might be due to a problem with the system itself not correctly aggregating bad values at the time they were recorded or the diagnostic triggers not being properly maintained themselves. Diagnostic tags were set up in the past by the team responsible for maintaining the database integrity to monitor critical system resources, but most of these monitoring systems failed intermittently and were thus unreliable for this study. This included valuable information on system uptime and service reboot or restart logging.
The potentially useful monitoring systems we found during this search are the memory usage tags described above and an external database table where buffer-store trigger times were registered. This database table was filled by a software trigger in the historian that is sent from the historian on every occasion of a network interrupt and is updated with the length of time the buffer was used. These database entries are not by definition a problem. The system already has measures to track this buffering. The database buffer entries can be used in an exercise to train a model to detect these same issues in the normal dataflow. In a fully defined monitoring system these events will be classified as 'non-emergency'. For the purpose of this research we will be using these trigger times as moments where data loss could have occurred in order to train a model, The monitoring framework that will result from this research will have a defined manner to add or redefine monitoring categories.

## 7.2. Data Selection

### 7.2.1. Database tags

For this project we will use data provided to us by a corporate partner. This is a dataset of a combined 4TB worth of datapoints spread over 7 databases collecting data from 14 sites spread over Europe and Asia. The oldest database stores data going back to 2005. This data was collected through interval polling by an intermediary server on each site to then forward this data to long term storage on the database. The databases in question each hold data on an median approximation of 4.000 tags per database up to approximately 12.000 tags for the largest databases.

Every tag in the system has a number of properties dependant on its type. For the purpose of this project we are only interested in a subset of these as illustrated in figure 7.1. These properties each might have an impact on the quantity of data being polled and thus the signature of the network data.

**Polling Frequency** The data collection for this particular system is driven by a polling system based on groups of tags. Depending on the specific collector the tags are polled by 1, 5, 30, and 60 second intervals.

**Max time interval** The maximum allotted time the system allows between datapoints. This setting makes it so the system registers whatever data it has in cache even if it does not extend past the deadband. On all datapoints we observe this period is set to one hour.

**Deadband** The database system works with a data compression algorithm that discards incoming values that do not significantly differ from the previous value.

**History** Each tag has a table with its associated historical data. These represent the indiviual datapoints for each tag.

The datapoints in the history each have the following fields:

**Timestamp** The time the value was registered. This can be backfilled by the the store-forward option on the intermediate server.

**Value** The value that was registered, or intuited, by the database engine.

**Quality** The database engine has an internal system for gauging the quality of the tag data. Good is unremarkable and means the data was correctly registered. Suspect means there was a problem with the handshaking but a value might be registered. Bad means the value was either not registered or was registered as an interpolated value as calculated by the boxing algorithm on the database.

Metadata can be retrieved from these tags by use of the 'aggregate' and 'history' functions provided by the database SQL engine.

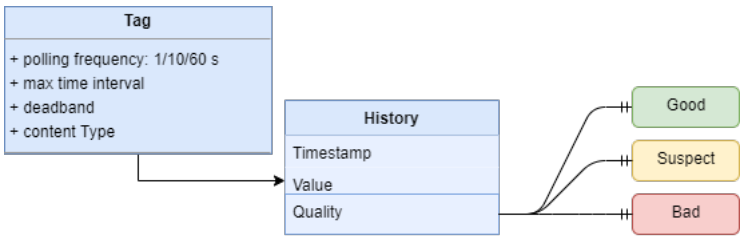


Figure 7.1: tag database architecture

The databases are also set up to monitor and store memory usage data. This monitoring was historically set up to detect memory leak issues that caused system failures by triggering a task on a set schedule that uses the .NET PerformanceCounter Dotnet-Bot to extract system resource information. The information gathered was recorded for specific database source connection processes that might impact the network data we will be monitoring and is therefore included in this study.

### 7.2.2. NETWORK DATA

The main focus of this study will be the network usage of the channel between the historian and the field. This channel is set up to poll data from an intermediate server physically located on the field to minimise permanent data loss due to network instability. This intermediate server then pushes the data to the historian if and when in normal operation. If the historian server is not available the intermediate server will send the data to an internal temporary database with limited storage to backfill the historian server when it becomes available.

To gather the network data, we need we set up a monitor on the intermediate server between the historian database and the field. The purpose of this server is to forward data to the historian but also to store data when a network issue occurs. The monitoring was set up as displayed in figure 7.2. Using the windows diagnostic PerformanceCounter we monitor the data throughput for each second by monitoring all TCP connections filtered on target server and process. This filter ensures we only capture data relevant to our process. The data gathered from this monitor was then written to a daily *.CSV file and stored locally on the monitored server. After 7 days of collection these files were then copied to a development environment for further processing.

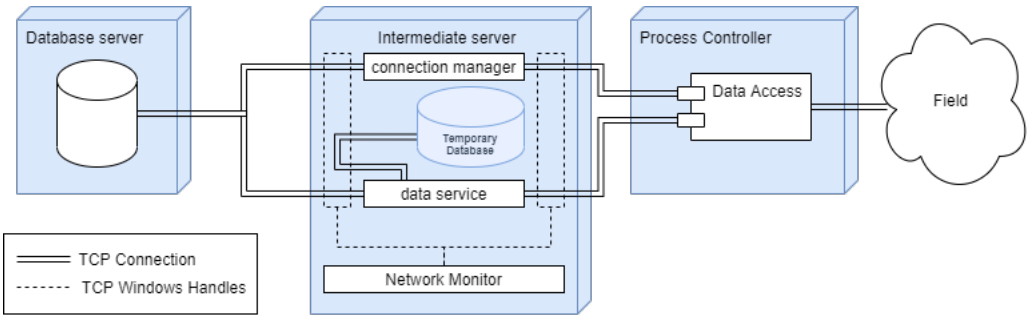


Figure 7.2: tag network monitor

### 7.2.3. DATA COLLECTION

On the development environment we used a combination of Python and C# .NET code to start processing the data. For Python we created the DataCollector class to group methods we would be using to collate all our data.

In figure 1 in the appendix we have the imports and global variables for the DataCollector class. The imported libraries will be discussed where they are used in the code. The global variables we use are a timestamp and an integer. The timestamp, filetime, exists to mark the output files with the correct date. The integer value, splitvalue, is used for paging sections of each file. The use of paging will be discussed later in this section.

In figure 2 in the appendix the network data was loaded into Python using the Pandas package. The Pandas package provides a powerful set of functions for processing datasets in the form of dataframes. The data we loaded from the CSV file was indexed on the 'Time' column. The time column was selected as index because it is a time series dataset where a temporal signature might be relevant to our question. After loading the data we use Numpy to split the dataframe into chunks small enough that the database can be queried for the additional data we want.

After multiple failed attempts to reliably use the ODBC datasource drivers for the historian database in Python, a data collector REST service was set up to execute the queries on the database and send this data to python. JSON was chosen as the best format to return this data as it is a standard in the industry for web communication.

The bad tag data is collected in the historian using the SQL query from figure 4. This query uses the inbuilt aggregation engine of the historian to return the number of bad tag values at every period interval.

The memory data is collected analogous to the bad tag data with a REST service. The SQL query for the memory data collects the interpolated data for all the memory tags that were saved to the historian database to track earlier issues. The data returned from these services is converted from JSON into a usable format. As part of this conversion the data is parsed through the function in figure 6 to convert the time column into a sortable datetime object.

This converted data is then linked to the networkdata on the time index and saved to a parquet file for fast recovery.

#### DATA UNDERSTANDING

The following parameters were selected based on the field experience of the individuals working with the system as possible indicators for issues. The points we chose to focus on are:

**BytesReceivedHistorian** The main datastream received from the data historian. This includes polling requests and configuration changes for a single DCS source.

**BytesReceivedDCS** The datastream received from the controller in the field. This is

mostly data from the monitored field systems, but can also include configuration data. item [BytesSentHistorian] The datastream sent from the intermediate server to the data historian. This is a repacked version of the data received from the DCS system.

**BytesInternalTraffic** The system circulates data through an internal service that allows for the creation of a buffer when a problem is detected. This problem detection is limited to a rule-based system and can therefore not detect all issues with the system.

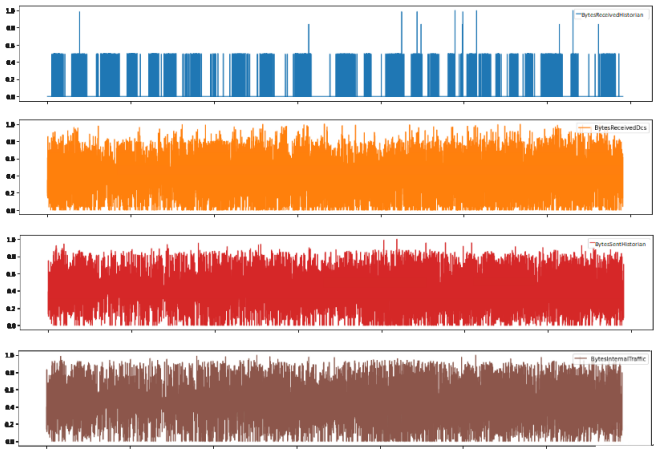

Figure 7.3: network data

**SUM(ng)** This is an aggregated count for the number of bad tags during the selected polling period. These are tags the system itself could identify as faulty, but cannot differentiate between system or mechanical issues.

**adlgp** This is a memory metric on the asynchronous data polling service.

**physmem** This is a memory metric for all memory in use by the data historian server.

**handles** This is a metric on all handles in use by the operating system on the data historian server.

From the data we have there is an expected high correlation between BytesInternalTraffic and BytesSentHistorian, as the internal traffic is the method the intermediate server uses to control the flow to the buffer that further feeds into the data sent to the historian. If these values ever diverge the result should be a drop of the datastream to the historian to zero or a noticeable increase in the datastream to the historian after such an event to backfill missed datapoints. A mismatch between these values where the datastream to the historian is not higher than the internal traffic or zero would be cause for concern.

After normalisation we can visually detect a deviation in the normal distrubution of BytesReceivedDCS where the lower 7 percent of values occur more often than a normal

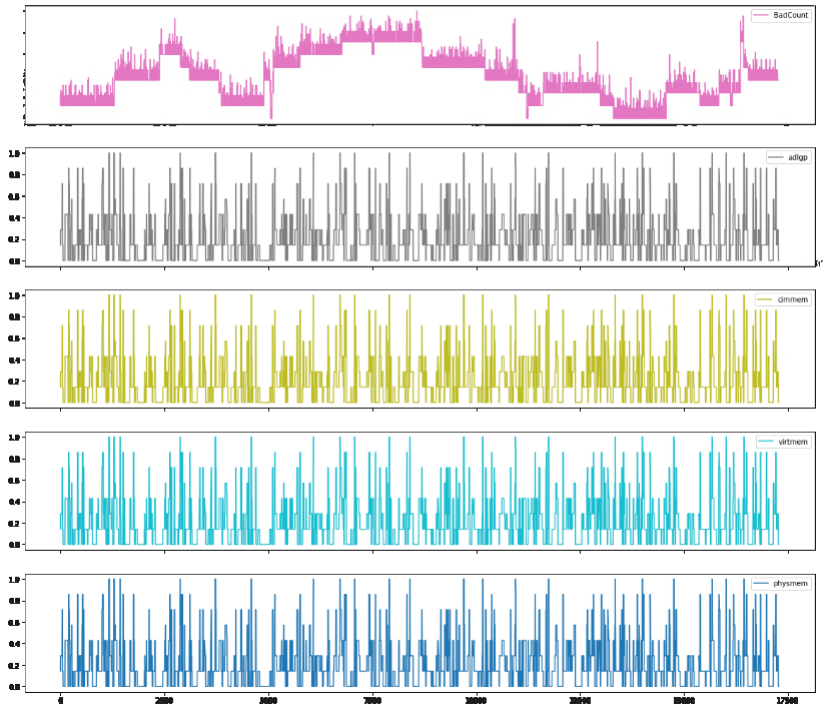


Figure 7.4: tag data

distribution would suggest. BytesSentHistorian and BytesInternalTraffic reflect this deviation but only the lowest 3 percent of values show a deviation from the normal distribution. This might be an effect of data compression or re-organisation in the intermediate server but can also be an effect of momentary data loss. The distribution of BadCount, adlgp, intermediate memory and historian memory do not have a normal distribution.

| | | Bytes Received Historian | Bytes Received Dcs | Bytes Received Manager | Bytes Sent Historian | Bytes Sent Manager | Bytes Internal Traffic | Bad Count | adlgp | cimmem | virtmem | physmem |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| data pre-normalisation | count | 17279 | 17279 | 17279 | 17279 | 17279 | 17279 | 17279 | 17279 | 17279 | 17279 | 17279 |
| | mean | 42.492042 | 26283.414058 | 0 | 716043 | 0 | 2509993 | 3629.991087 | 1.46542 | 1.46542 | 1.46542 | 1.46542 |
| | std | 119.358885 | 12316.861614 | 0 | 292739 | 0 | 1003412 | 9.116259 | 1.598029 | 1.598029 | 1.598029 | 1.598029 |
| | min | 0 | 0 | 0 | 0 | 0 | 0 | 3615 | 0 | 0 | 0 | 0 |
| | 25.00% | 0 | 21302 | 0 | 619380 | 0 | 2179916 | 3622 | 0 | 0 | 0 | 0 |
| | 50.00% | 0 | 25079 | 0 | 706439 | 0 | 2474439 | 3630 | 1 | 1 | 1 | 1 |
| | 75.00% | 0 | 32549.5 | 0 | 829060 | 0 | 2878191 | 3635 | 2 | 2 | 2 | 2 |
| | max | 760 | 65373 | 0 | 1712141 | 0 | 5546333 | 3657 | 7 | 7 | 7 | 7 |
| data post normalisation & outlier removal | count | 17279 | 17279 | 17279 | 17279 | 17279 | 17279 | 17279 | 17279 | 17279 | 17279 | 17279 |
| | mean | 0.055911 | 0.402053 | 0 | 0.418215 | 0 | 0.45255 | 0.356931 | 0.209346 | 0.209346 | 0.209346 | 0.209346 |
| | std | 0.157051 | 0.188409 | 0 | 0.170978 | 0 | 0.180915 | 0.217054 | 0.22829 | 0.22829 | 0.22829 | 0.22829 |
| | min | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | 25.00% | 0 | 0.325853 | 0 | 0.361758 | 0 | 0.393037 | 0.166667 | 0 | 0 | 0 | 0 |
| | 50.00% | 0 | 0.383629 | 0 | 0.412606 | 0 | 0.44614 | 0.357143 | 0.142857 | 0.142857 | 0.142857 | 0.142857 |
| | 75.00% | 0 | 0.497904 | 0 | 0.484224 | 0 | 0.518936 | 0.47619 | 0.285714 | 0.285714 | 0.285714 | 0.285714 |
| | max | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |

Figure 7.5: normalised data statistics

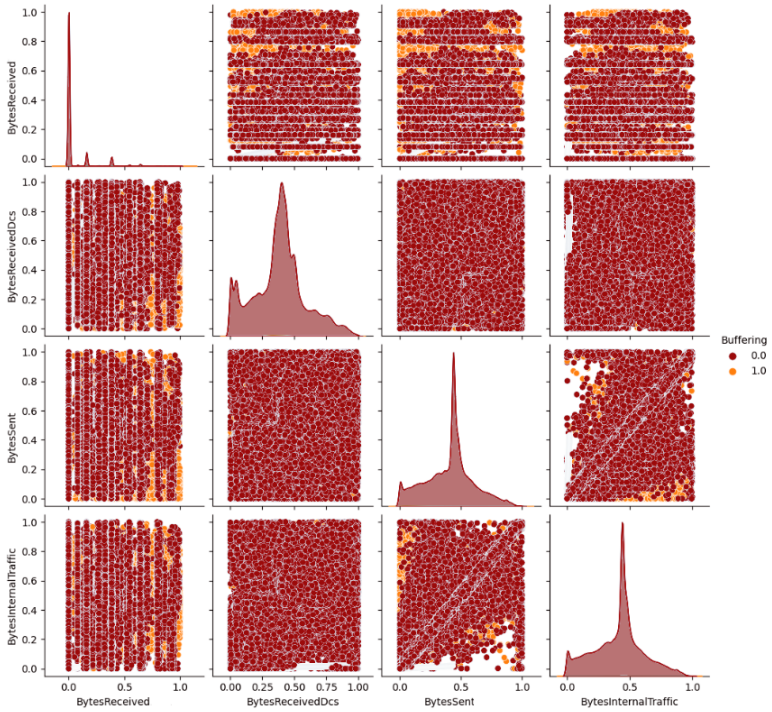


Figure 7.6: comparing and plotting

In figure 7.6 we plotted the main features for our data to detect patterns in the ratios between these features. Something that is noticeable in these plots is the points where Buffering is 1, the classification we use to detect issues, is often found on the edge of the plots.

### DATA DISTRIBUTION

From our data we generated boxplot for the dataset as a whole in 7.7 and specifically for the data where it has been classified as bad in 7.8.

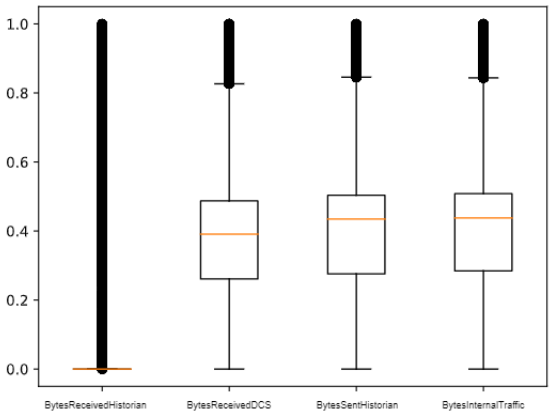


Figure 7.7: complete dataset boxplot

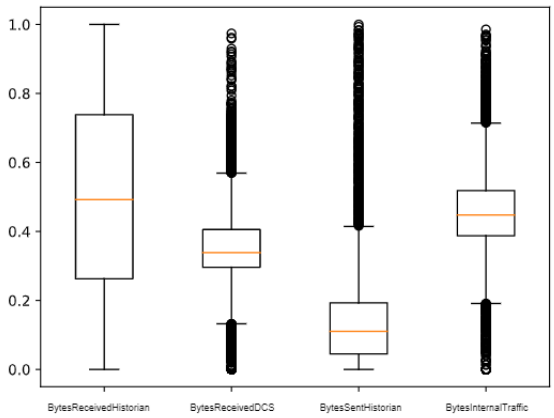


Figure 7.8: positive classified dataset boxplot

From these figures we can deduce a number of differences between the classes of data. The full dataset has a BytesReceivedHistorian distribution that varies wildly. Something that can also be seen in 7.3 where data spikes indicate data is not streamed but sent in bulk transfers. The other features are more evenly spread with some outliers still present after normalisation.
The boxplot for the bad data shows a distincly different distribution. The BytesReceived-Historian shows what appears to be a normal distribution. The other features have more outliers but do still show data is being sent and received. This behaviour is what prompts the research into the use of Machine learning as a simple rule based system throws a lot of false positives in this scenario.

#### NORMALIZATION

We normalised the data by removing outliers more than 3 standard deviations from the norm and replacing these values with an average value to maintain the integrity of the time series nature of the data. After this stem we applied a minmax fit to the set.

### 7.2.4. PRINCIPLE COMPONENT ANALYSIS

Using the Sklean decomposition PCA algorithm and maintaining a 0.95 fidelity to the original dataset we downsampled the dataset to 4 points. VAR modelling gets results that indicate an optimal lag prediction of 16 time intervals.

### 7.2.5. TIME SERIES PREPARATION

To increase the time series characteristic of the data we formatted the data into overlapping windows. This sliding window creates a 5 by 4 input shape for the model that can be used to compare a 5 second signature for the features that are being observed as can be seen

visualised in figure 7.9.

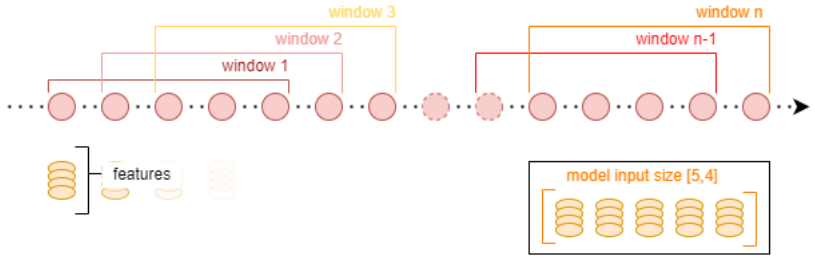


Figure 7.9: sliding window selection

### 7.2.6. MODELS USED

For this research we have selected a number of different models to compare their detection capabilities. In the following subsection we will discuss the models we researched.

#### LINEARSVC

The Linear SVC is an implementation of a Support Vector Machine with the weighted parameters added together to form a single classification value. The support vector machine was first introduced at Bell laboratories in 1992 [Boser et al., 1992] The concept of a support vector machine is built on the usage of hyperplanes. These hyperplanes are generated in a feature space without set coordinates using an inner product calculation to create a gram matrix as visualised in figure 7.10 .

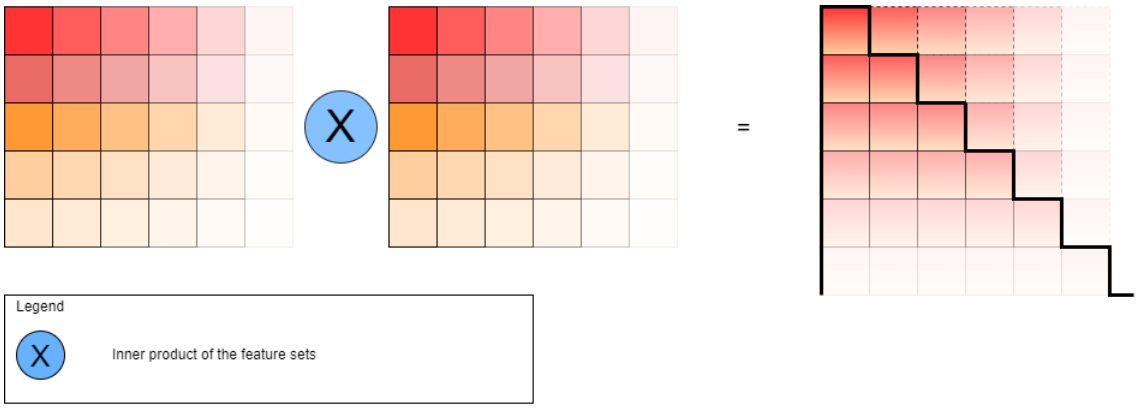


Figure 7.10: creation of a gram matrix

This concept of using inner product calculations is called the 'kernel trick' and removes the necessity of calculation coordinates for each feature making the entire process computationally cheaper. Based on these gram matrices weights can be defined to maximise the difference between categories.

A support vector machine can then be trained and used using either a soft or hard margin. With the hard margin the resulting classification is either true or false. A soft margin

can give a gradient of truth based on the distance the input value is from the hyperplane and allow for the manipulation of a threshold value for truth that can be used to compare to other classification techniques or later finetuning once the model is in operation. The manner in which the hyperplane classsifies the datapoint can be seen in the sequence of figures in 7.11

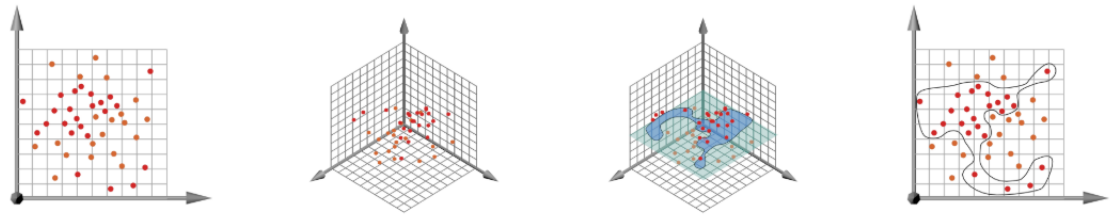

Figure 7.11: hyperplane classification sequence

### NEURAL NET

Because we will use multiple types of neural networks in this section we will first discuss the general workings of a neural network and in the next sections we will go more into detail on specific types of neural network we will be testing.
The concept of a neural net is based on the functioning of a biological nerve cell see figure 7.12 as an example. In a biological neural system a large number of cells are interconnected and send impulses to other cells in the network. Based on feedback the cell receives in the form of a reward or punishment the system will be formed in such a manner as to optimise for prefered results.

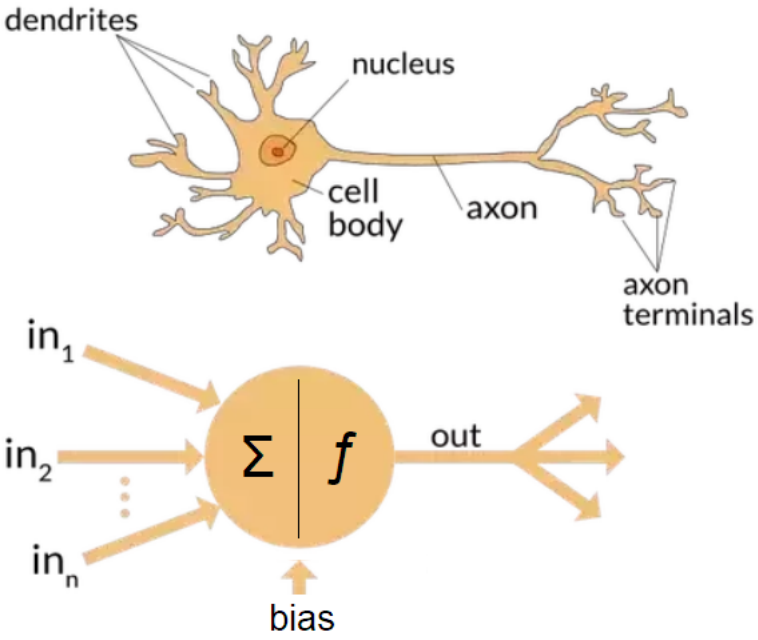


Figure 7.12: biological neuron vs machine learning neuron as found on [Nagyfi, 2018]

The machine learning neural net is likewise consisting of multiple nodes working together to learn and optimise for the classification of data. A neural net works by grouping a number of nodes in a layer, sending data to this layer and then forwarding the output of the nodes in this layer to a next layer. After a number of these layers the output of the last layer is sent to a single result node that returns a classification of the input data. This output can then be compared to a known classification and can send feedback to the underlying layers. Each node in the underlying layers will have given a classification on what the eventual output will be based on a bias value to have variety amongst the nodes in the system and a weight which the system as a whole can use to determine how much impact the node will have on the overall result. The feedback from the known classification can decrease the weight of a node to make its input less relevant in the whole, this creates a system where connections that correctly classify a datapoint are reinforced.

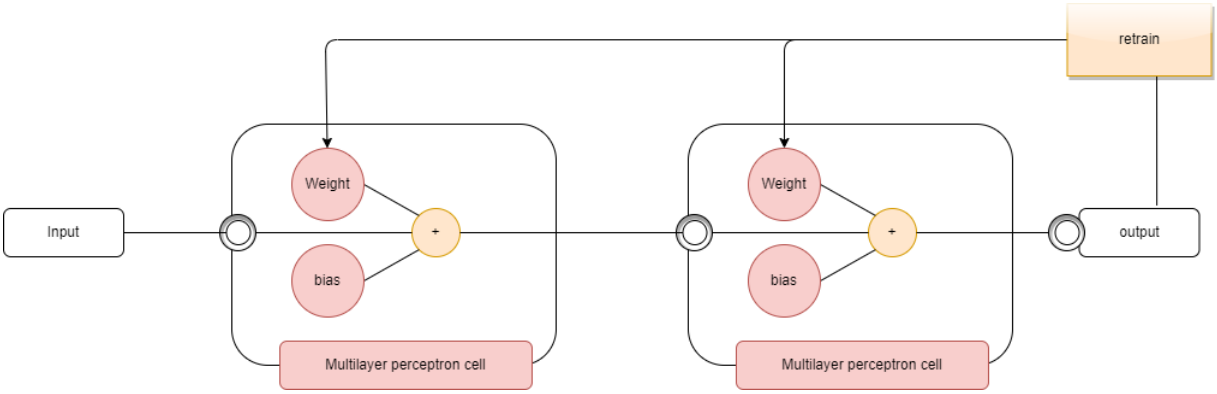


Figure 7.13: simple mlp network

### Convolution in neural nets

Convolution in mathematics is an operation on 2 functions that produces a third function that expresses the overlap of on function as it is shifted over the other. The result is a type of blend between these functions. This operation can then be used in a neural net to convolve the input of each layer with a filter and send the result to the next layer. This filter can be adapted through the machine learning process to better fit the required output. Each neuron in the layer only processes data for a subsection of this result set, also known as its receptive field, although fully connected layers are also possible. Convolutional neural nets may also include pooling layers. These pooling layers are used to reduce the dimensionality of the results.

### LSTM

The long-short-term-memory node is a neural network that uses a specialised type of node. An LSTM is a specifically good model to use for time series data. It is an improvement on a standard RNN by extending the functionality of a node by using a forget gate by [Gers et al., 1999] that also describes the strengths of RNN and LSTM specifically. The forget gate was added to solve the vanishing gradient problem. The vanishing gradient problem is an issue standard RNN's have where a node's weight can decay to zero. The LSTM resolves this issue by implementing a forget gate that checks the input values and previous cell state to prevent a weight change that would lead to a zero weight. The LSTM forget gates work by sending the information from a hidden state and the current input through a sigmoid function. The result of this function is then multiplied with the long term state to use as a parameter in the classification of the new datapoint. We chose it because of its specialisation in time series data.

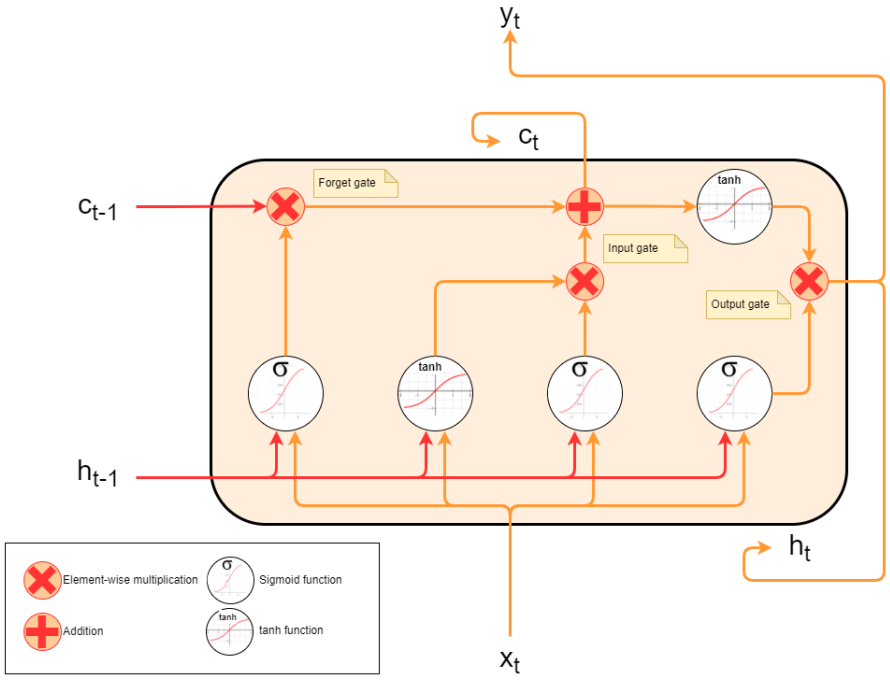


Figure 7.14: the construction of an LSTM cell

### WAVENET

Wavenet is a neural network that was developed for use with Text-to-speech technologies. Although the data we use is not audio, we used a simplified wavenet algorithm to model our data based on the idea that audio data is in essence also time-series data. Wavenet is a causal linked dilated convolution. This forces the model to work sequentially, working sequentially allows the model to recognise failing pattern that are time dependant. The dilation in the convoluted nodes increases the models performance as it can cover a larger sequence of datapoints without the pooling cost a standard convoluted neural net has. Having a causal linked dilated model give the model the ability to detect patterns with various levels of granularity. We chose to include our simplified wavenet implementation to contrast against the LSTM. Where The LSTM model has more complex nodes, the simplified wavenet has a more complex network layout. Comparing these would give us a wide spectrum on modeling techniques available to time series modelling.

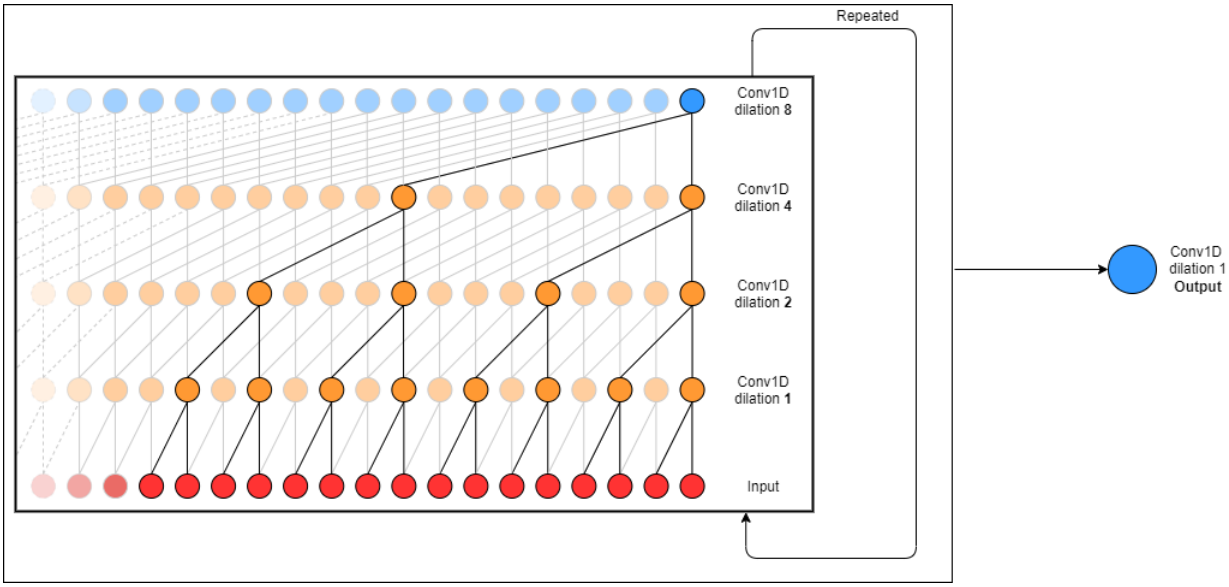


Figure 7.15: the construction of a wavenet layered model

### WAVENET -LIGHT

as an experiment we also implemented a causal linked dilated convolution followed by a GRU node. This GRU node uses recurrence in the output of the wavenet implementation to finetune the final result of the model.

### 7.2.7. SOFTWARE

To implement our models in Python we used the keras libraries. These libraries are an abstration over Tensorflow and include a wide variety of functions and predefined layer types to create our models. This abstraction, building a model with neurons and layers makes it easy to work with, without having to know the full detail of the underlying mathematics. We did not use Pytorch because Keras provided a beginner friendly environment with general functions and object pre-assembled whereas Pytorch is more finely grained and thus provides a steeper learning curve.
We chose Python as a programming language because of its versatile use in both the world of software development as its extensive use by academia where it has proven its value in creating mathematical models.

## 7.3. SUMMARY

In this chapter we discussed the scope of the data and environment we work with, we have a large quantity of historical data to work with and a smaller quantity of network monitor data to link with. We have determined some features to be directly linked to others and have determined some to be insignificant or unchanging even when problems occurred. We have determined that for the purpose of this research we can work with a limited section of data that after normalisation reduces down to 4 features we can use to train our models. Other features however cannot be entirely excluded in follow up research as different types of event might be categorised by using these additional features.

# 8

# MODEL EVALUATION

## 8.1. DESCRIPTION

To detect our events we created multiple models using different techniques. We are attempting to create a model based on time series data, this shaped the choice of techniques we considered in this research. The techniques we used and compared are linear Support Vector Classification, Long-Short-Term-Memory neural network (LSTM for short), and a simple implementation of a wavenet algorithm. The models will be compared by means of an ROC curve and the corresponding AUC value and the confusion matrix created by setting an optimal threshold based on the precision recall curve.

### 8.1.1. DEFINING SUCCESS PARAMETERS

The models we are using need to be evaluated by a set of metrics. We will be using precision and recall to define the threshold for our positive measure. Precision and recall are according to [Handelman et al., 2019] amongst other metrics used in this paper a common method to define a threshold for machine learning models. We specifically chose this method of determining a threshold to maximise the ability of our system to detect problems whilst keeping the false positives to a minimum.

The rationalisation behind this is that up to a few minutes of interruption does not heavily impact the system, but more than a few false positives will lead to the responsible engineers ignoring alerts. In the graphical representation we use precision and recall are mapped against a possible threshold value that can be used to classify a true or false result.

This threshold starts from 0 where all results are classified as true up to a maximum of 1 where all results have been mapped. precision is the ratio of true positive results against the total number of results that are classified as true according to a specific threshold. The

graphical representation of the precision we show maps this ratio to all possible thresholds. The curve on this graph can fluctuate and move back down after it has peaked because precision is a ratio and if the number of true positive results rises slower than the number of false positives the curve will start dipping down again.
Recall is the ratio of true positive results against the total number of classified results. The total number of classified results is the sum total of all true positive and false negative results that were classified using this threshold. The curve this graph makes should start in 1. If this curve starts in 0 this would mean the model under observation classified a false negative as the most likely true value and this would be a bad model.
The recall curve bottoms out at the ratio where all results are classified as true. In contrast to the precision curve, the recall curve will only slope down.

For the purpose of this research we will always try to maximise both precision and recall. To achieve this objective we will determine the threshold we use for classification of a model as the threshold at the point of intersection between precision and recall on the graph.
Besides the precision recall curve we will also be using an ROC curve to visualise the effectiveness of the models.

The ROC curve (receiver operating characteristic curve) plots the threshold with true positives against false positives. The curve this generate is a good indicator for the effectiveness of a model. A perfect curve would move up horizontally until it reaches the maximum value for true positives and then follows in a right angle. A diagonal line that follows from (0,0) to (1,1) indicates a curve where the classifier works exactly as well as random chance. From this follows that the AUC (area under the curve) a good indicator for effectiveness is. An AUC of 0.5 or less indicates a classifier that works as well as random or worse. An AUC of 1 would indicate a perfect classifier.

### 8.1.2. Evaluating the success of a model

The success of a model will be based on the output from a Confusion matrix. The confusion matrix for our dataset plots 4 quadrants comparing the classification of the output of a model to the known classification. The quadrants in the confusion matrix set the number of true positives, true negatives, false positives and false negatives against each-other to visualise the quality of the output generated by the model. For the purpose of this research a false negative would be a more impactful problem then a false positive. A false negative could lead to a smaller number of important tags failures to stay unnoticed. A false positive would lead to an engineer wasting a minimum of time to trace the veracity of a warning signal. The most important factor in false positives would be the responsible team possibly ignoring true positives. The threshold for this would have to be indicated by usage.

### 8.1.3. Linear SVC Configuration

The Linear SVC is a standard implementation of a Support Vector Machine with the weighted parameters added to form a single classification value. A Support Vector Machine works by generating a hyperplane vector that functions as a divide between categories. It is a general and reliable machine learning technique that can be used to model a variety of datasets. It is also a computationally inexpensive technique and was chosen because it is a good baseline to compare other more complex modelling techniques to. In our implementation we use the CalibratedClassifier in Keras on top of the Keras linear SVC implementation to generate a probability of a time point being problematic. This gives us more finely tuned control over what threshold to define as the correct classification and gives us a more direct analogue to the other models we run.

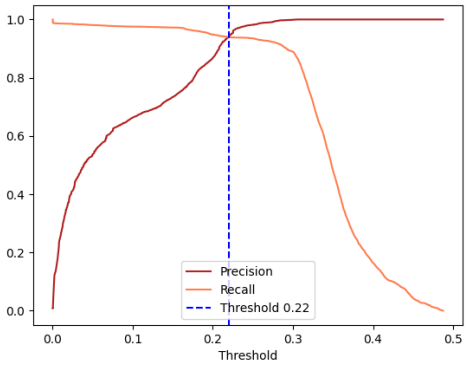


Figure 8.1: LinearSVC Precision/Recall graph

For the events we predetermined to be divergent, the Linear SVC model performs well with a AUC score of 0.9997. This score is the highest of the models we tested and could be an indication for it being the better method. The flattened curve indicates a high accuracy, but in the edge cases detection becomes more difficult and gives more false positives. This shape can also be explained by the Linear SVC model correctly identifying a true negative most of the time. Due to the rarity of the events in the overall data this gives a slightly distorted view.

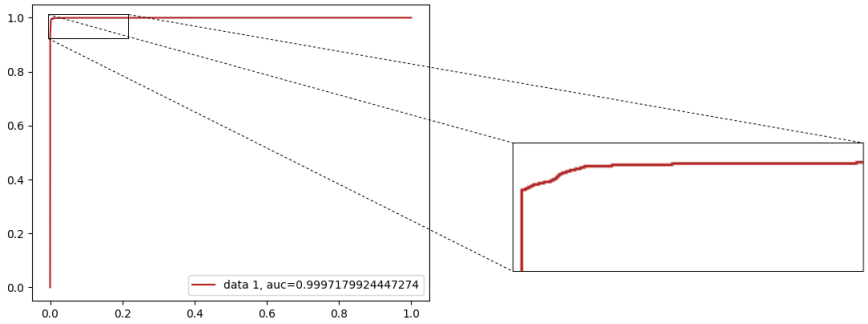


Figure 8.2: LinearSVC roc with zoom

### 8.1.4. LSTM CONFIGURATION

For the LSTM configuration we use a 3-layer model composed of an input layer with 30 nodes, a hidden layer of 30 nodes and a single dense layer to deliver a classification prediction. The input layer receives a 5 by 6 input matrix to analyse 6 variables over a 5 second moving window. We ran this model with a batchsize of 10 for up to 50 epochs. When a moving average of 2 minutes is applied to these predictions we can set a precision-recall curve to determine the optimal threshold. This curve is displayed in figure 8.3 The precision and recall curves meet at the threshold of 0.1 which gives us a recall and precision of 95 percent each. For the events we predetermined to be divergent, the Linear LSTM model performs

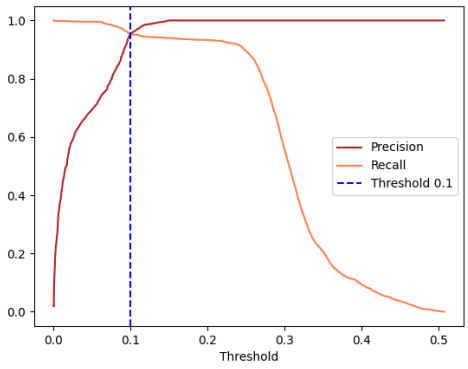


Figure 8.3: LSTM Precision/Recall graph

well with a AUC score of 0.9995. This score is lower than the SVC model but within our selection margins for preference. When selecting for a more time balanced dataset where a larger portion of the data is event data the LSTM model more closely follows the correct classification as can be seen in the appendix picture 11.

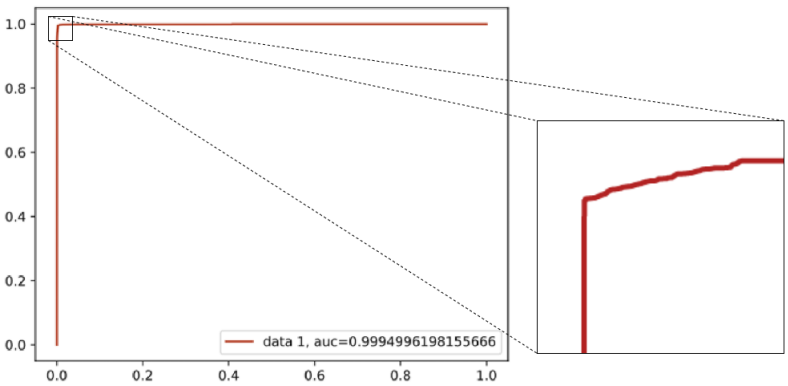


Figure 8.4: lstm roc with zoom

### 8.1.5. WAVENET CONFIGURATION

The model we use in this research is a model that uses 4 convoluted layers with a kernel size of 5 followed by a Keras time distributed dense layer to condense the convoluted layers

down to a single return value.

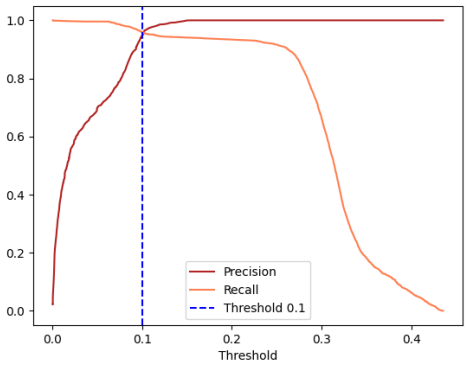


Figure 8.5: Wavenet Precision/Recall graph

For the events we predetermined to be divergent, the Wavenet model performs well with a AUC score of 0.9996. This score is lower than the SVC model but within our selection margins for preference.

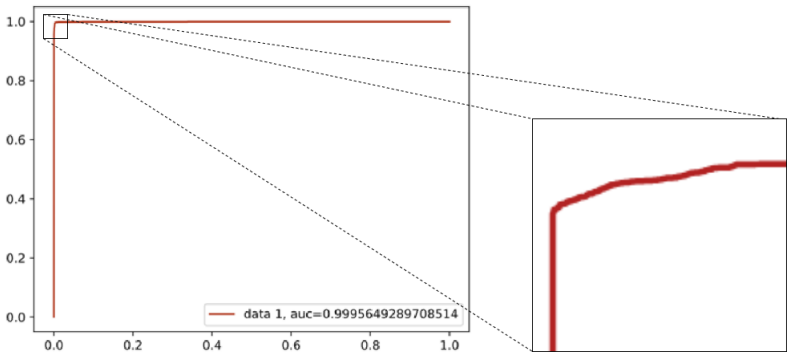


Figure 8.6: Wavenet roc with zoom

### 8.1.6. Model Predictions

Using the threshold suggested by the precision recall graph gives us the confusion matrix in figure 8.7 for the svc model. From this confusion matrix we can see most results can be correctly classified as either true or false. With a threshold set at 0.22 from a total of 787727 datapoints 787033 were correctly classified as either normal or abnormal. The total number of datapoints that were classified incorrectly either way is only 694. This accounts for only 0,09 percent of all cases and even compared to only the true positive cases false labeling only account for 0.111 percent of the total of these cases.

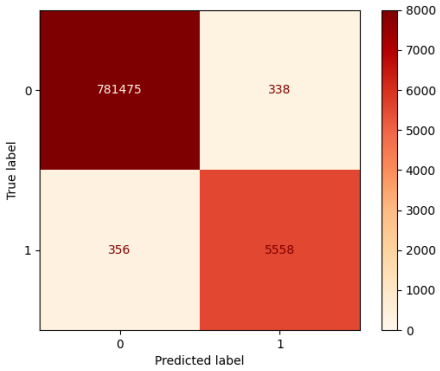


Figure 8.7: Linear SVC Confusion Matrix with threshold 0.22

Using the threshold suggested by the precision recall curve gives us a the confusion matrix in figure 8.8 for the LSTM model. From this confusion matrix we can see most results can be correctly classified as either true or false. With a threshold set at 0.1 from a total of 787727 datapoints 787165 were correctly classified as either normal or abnormal. The total number of datapoints that were classified incorrectly either way is only 562. This accounts for only 0.07 percent of all cases and even compared to only the true positive cases false labeling only account for 0.09 percent of the total of these cases.

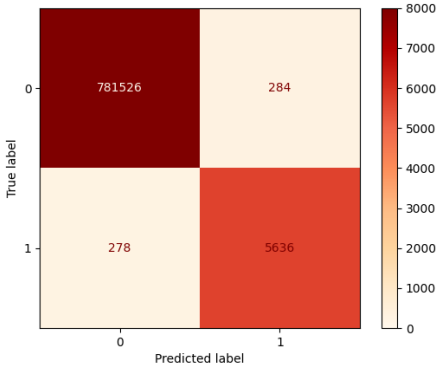


Figure 8.8: LSTM Confusion Matrix with threshold 0.1

Using the threshold suggested by the precision recall curve gives us a the confusion matrix in figure 8.9 for the Wavenet-light model. From this confusion matrix we can see most results can be correctly classified as either true or false. With a threshold set at 0.1 from a total of 787727 datapoints 787172 were correctly classified as either normal or abnormal. The total number of datapoints that were classified incorrectly either way is only 555. This accounts for only 0.07 percent of all cases and even compared to only the true positive cases false labeling only account for 0.09 percent of the total of these cases.

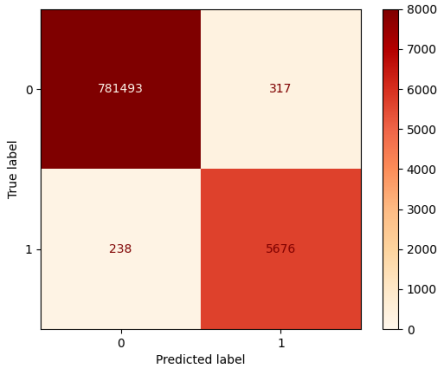


Figure 8.9: Wavenet Confusion Matrix with threshold 0.1

Because of the sequential nature of the data we also made a visual observation of the data. In figure 11 in the appendix we trended the predicted classification, the real classification and the trained classification. For the event window shown in this figure we can see that the moving average classification almost completely overlaps with the training event. There is also a lower value leading prediction. For 20 minutes before the classified event in the test set the model gives indications of a higher probability that does not meet the threshold of the event. All models provided similar results but the Wavenet implementation gave a result that more closely matches our event classification over the other 2 methods under observation.

# 9

# ARCHITECTURE

This project was implemented using a combination of program languages and technologies broken down into sections with their own responsibilities and challenges. For data acquisition there is a service running on the intermediate server that asynchronously pulls data from windows management handles. This data acquisition service compares the total byte transfer counter for a predefined set of network traffic handles for each second. This function must work asynchronously to exclude external factors from the timing or data measurements. To easily use Microsoft Windows handles we chose to use a C sharp .NET program that provides access to Windows handles.

The data collected by this service is sent to an analysis service with 30 second intervals. Another data acquisition service runs on the data historian server where the analysis service will be able to poll for quality information over a 30 second interval with 1 second interpolation. Combined this data can then be fed into the model to predict problems. If the analysis service detects an issue it can attempt to mail a technician.

If these intermediate servers exist on a network with multiple similar systems, a module could also be includes to monitor its peers on the network. Every preset time interval the services broadcast a randomly generated number to its peers. Depending on a predefined ruleset, if one of these services does not reply the service with the highest generated number will send a notification mail to a technician and a signal to its peers.
The server that could not be connected to will then be discarded from the signal pool for a set duration before a recheck must occur. The server with connection issues can only detect itself so has the highest number and will also send a notification. If this notification mail arrives the technician can interpret this as a horizontal issue where intermediate server communication is out, but vertical communication might still be operational.

The analytical model itself will run in a separate Python-based service. A retraining service allows for the training of a new model and because the model runs in a standalone

service it can be detached and replaced without any breaks in the system as a whole.

## 9.1. SERVICE COMPOSITION

The collection of microservices constituting the data loss monitoring application for a single system can be visualised as in 9.1. This is an overview of how the various services would interact on an intermediate server. It also shows both the need for the microservice architecture through the use of multiple programming languages and the abstraction of the monitor task to increase expandability to additional datasources.

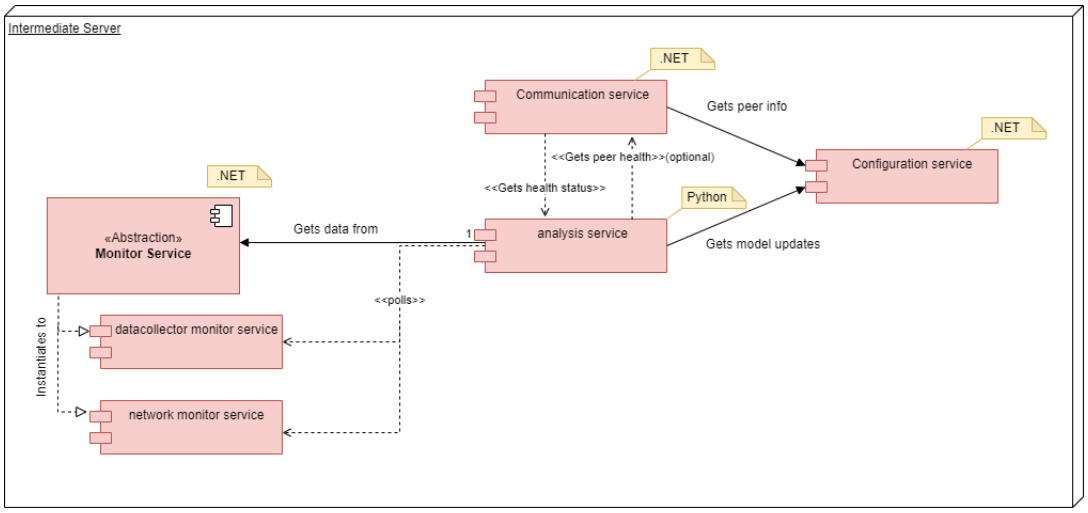


Figure 9.1: monitor microservice architecture

### 9.1.1. MONITOR SERVICE

The monitor service, as visualised in 9.2 implements an abstraction layer that removes the datasource specific code from the operation of the polling system in the program itself. This allows us to create different versions of the monitor manager that all work by polling data and adding the collected datapoints to a queue to be retrieved without the need to rewrite or copy code. The only method that has to be overridden is the polling method itself and this can be done with minimal effort in case an additional datasource version needs to be created.
The polling will run on an asynchronous thread to prevent delays in the execution of the software from influencing the monitoring frequency.

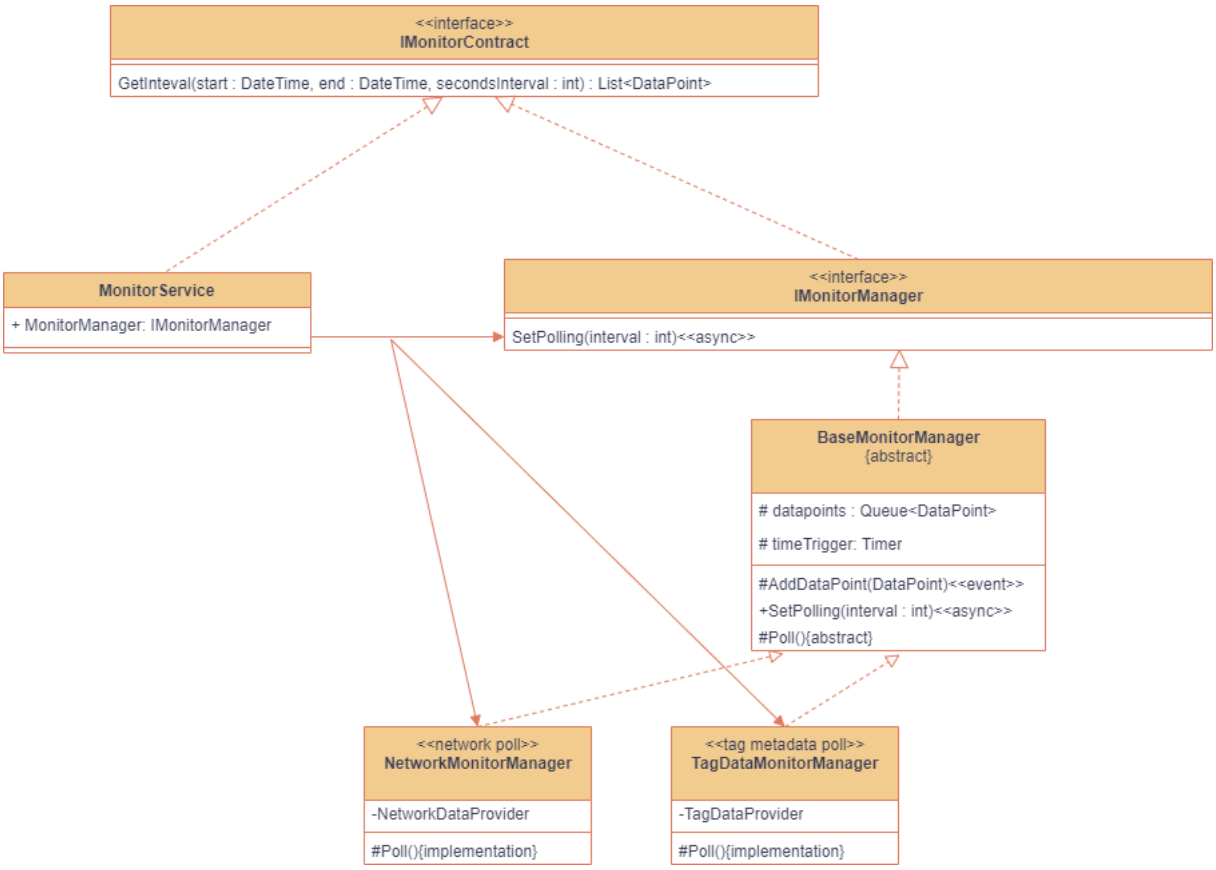


Figure 9.2: monitorservice UML diagram

### 9.1.2. COMMUNICATION SERVICE

The communication service, as visualised in 9.3 is the channel the application will use to notify external systems or users of any problems. It implements both a peer health check where it polls all the peers its configuration contains and periodically checks with the analysis service for the current health status of the data is is monitoring. The use of a set configuration file where peers are registered, opposed to a broadcast system is to prevent intruders from impersonating a peer to receive information.
If the communication service detects any problems it will notify through email or a watchdog tag.

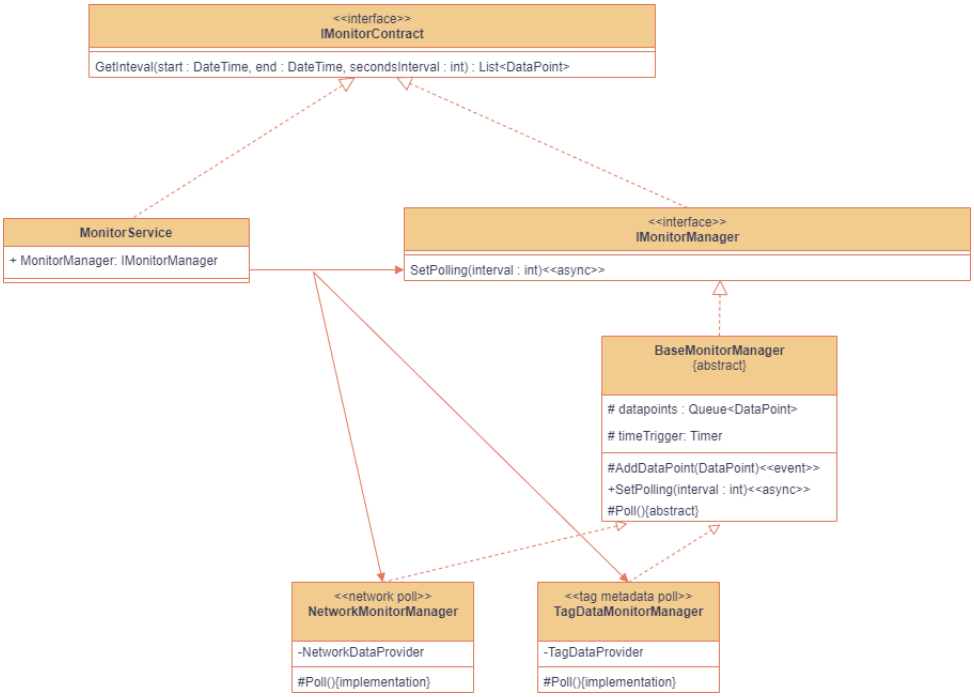


Figure 9.3: communicationservice UML diagram

### 9.1.3. ANALYSIS SERVICE

The analysis service, as visualised in 9.4 implements the execution of the analytical model that will run and classify the polled data. This is the only service that will be created with python because of its specialisation in mathematical modelling.
The analysis service will retrieve data from the monitor services and merge this data to get a feature set it will use to generate a classification. It will hold this classification in a queue. This queue will be polled by the communication service.
We specifically chose for a polling architecture as opposed to a publish/subscriber pattern because the polling method can be more easily synchronised and has a built in detection method for failing services. It is easier to poll and get an error code than waiting for an event that will not come.

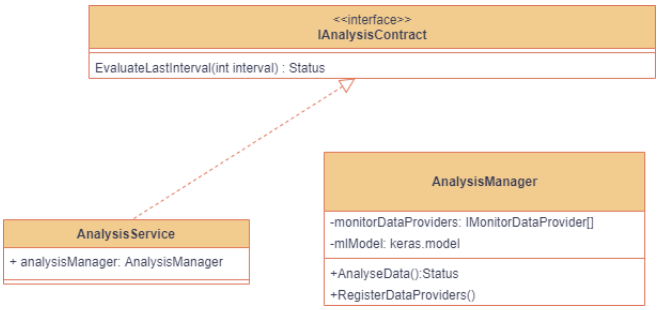


Figure 9.4: analysis service UML diagram

### 9.1.4. CONFIGURATION SERVICE

The configuration service, as visualised in 9.5 implements access to a configuration file. This single access point to the configuration provides the list of peers the monitor service can use and the optimal detection frequency and most recently trained model for the analysis service to use to keep its detection up to date.

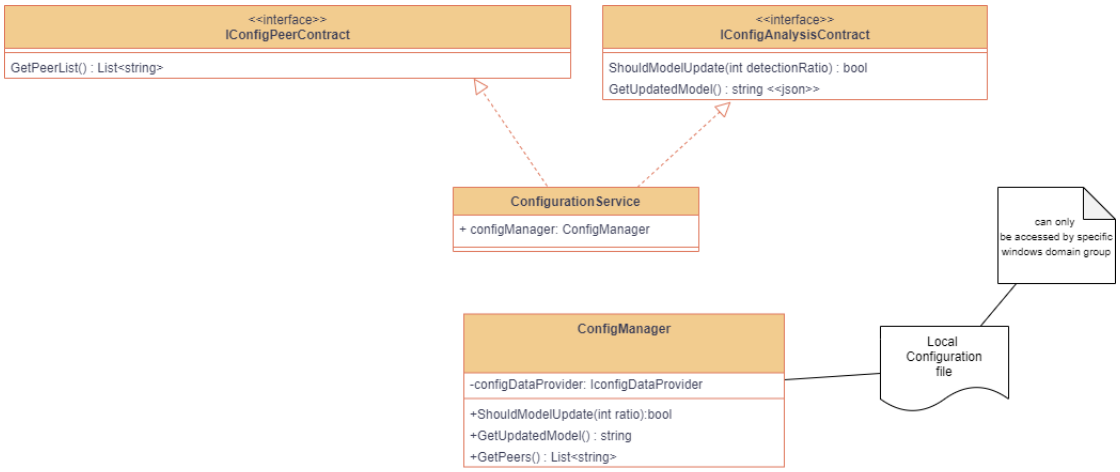


Figure 9.5: configuration service UML diagram

## 9.2. SUMMARY

The UML schemes in this chapter describe the setup of a single monitoring node. They were designed to optimally take advantage of the microservice architecture and would run as a robust system to both detect problems in network traffic on a node and amongst its peers on the network.

# 10

# Conclusion

From this research we can conclude the following answers to our research questions.

- Rq1: How can recorded data from a data historian be linked to significant differences in network usage?

  When analysing the data we could not find any links between meta data in the historian and network usage. Possible reasons for this could be that the problems we detected were always buffering events. We will need to continue this research to include events of different types to be able to expand our list of failure events. The hypothesis for this lack of registration in metadata is that the database will backfill the buffered data which means we lose this valuable meta information.

- Rq2: What algorithms are best to teach machine learning to monitor categorised data usage?

  From the models we used, all were very successful at detecting the events we had access to. The more complex models were slightly better in that they more closely followed the defined classification. The least complex model, the SVM, actually had a higher AUC and for the data and event types we could work with was more than sufficient to classify our events. The SVM is also faster to train than both the LSTM and Wavenet models. This would be an advantage when trying to retrain the model to be more accurate but when the data becomes more complex it might fall behind.

- Rq3: What is the best architecture to create an alert system to warn of a pattern drop and integrate a ML subroutine?

  The Keras model we will use for the analysis of the data itself can be replaced at runtime in our code by virtue of the abstract manner it was implemented. This also pushes us to a microservice architecture because of

the use of python for this specific part of the application.
Alerting was implemented as a peer network that would alert to a problem even if the monitoring service itself becomes unresponsive. This provides a highly reliable system without a single point of failure that could prevent a problem from being detected.
The data collection service was abstracted to allow more data to be collected from different sources. This allows for a high level of maintainability and extensibility.

- Rq4: What are the requirements to maintain a machine learning model in a microservice structure?

  In the manner we implemented the machine learning models we can use the deserialisation functions provided by Keras. We can use this function to update the model at runtime. This new model can be trained on a development machine and be uploaded to the configuration service to be used in the application. This even allows us to switch out the model for an entirely different structure.

From this we can distill the answer to our original question:

> Can an AI microservice infrastructure be created to monitor network data and compile this data into a message to peers on a network to generate alerts when a pattern drop is detected?

The architecture we propose in this paper is capable of something similar which serves the same purpose. It can detect problems with the network communication and send an alert when problems are found. When multiple peers exist it can also alert to failures in communicating with its peers. This is not exactly the same as the original question posed because if a service can detect a problem but cannot contact its peers it cannot send an alert message. Its peers however will send an alert because the service node with an issue cannot be contacted.
For the events we could label, the optimal model was the wavenet model. This is in line with what could be expected from related work, as this model was designed with time-series data in mind. However the other models were close in effectiveness.
When implementing this detection improvement in the system it is hard to make a direct cost-benefit analysis. Most of the benefits from this work would be indirect in the creation of more confidence in the data that is being gathered. The ephemeral nature of this benefit is impossible to distill into hard numbers but might become clearer with more work on the subject.

## 10.1. Future work

- The effort in this thesis highlighted the need to categorise bulk data and can with the benefit of hindsight be repeated with a larger collection of data that is also labeled in

instances where backfill occurred. This could give more accurate results and could also serve a blueprint for how to label time series data as it is used in the industry.

- Another interesting avenue we noticed when working on this research is how closely we could have a machine learning model follow live data. This could lead into research that would create a machine learning clone for incoming data that would help improve industrial operations by optimising certain processes to better fit an ideal or create a shadow record that would give a result similar to the original data in case of dataloss.

- Something that could also be interesting to research is a cross comparison of different programming languages and paradigms on processing architectures for various types of machine learning models.

The potential for further research on the topic of data loss and system health in industry is great. From experience many industrial sites keep their operational network closed for research. This guarded nature sets limits on what can be accomplished with this data. More cooperation between academia and industrial partners could help both increase understanding in this area.

# BIBLIOGRAPHY


Specification of Internet Transmission Control Program. RFC 675, December 1974. URL https://www.rfc-editor.org/info/rfc675. 16

Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181. 13

Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, page 144–152, New York, NY, USA, 1992. Association for Computing Machinery. ISBN 089791497X. doi: 10.1145/130385.130401. URL https://doi.org/10.1145/130385.130401. 32

Robert T. Braden. Requirements for Internet Hosts - Communication Layers. RFC 1122, October 1989. URL https://www.rfc-editor.org/info/rfc1122. 16

L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984. ISBN 9780412048418. URL https://books.google.be/books?id=JwQx-WOmSyQC. 11

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001. ISSN 1573-0565. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324. 11

Laizhong Cui, Shu Yang, Fei Chen, Zhong Ming, Nan Lu, and Jing Qin. A survey on application of machine learning for internet of things. *International Journal of Machine Learning and Cybernetics*, 9(8):1399–1417, Aug 2018. ISSN 1868-808X. doi: 10.1007/s13042-018-0834-5. URL https://doi.org/10.1007/s13042-018-0834-5. 17

Ibrahim Ali Ibrahim Diyeb, Anwar Saif, and Nagi Ali Al-Shaibany. Ethical network surveillance using packet sniffing tools: A comparative study. *International Journal of Computer Network and Information Security*, 10(7):12, 2018. 15

Dotnet-Bot. Performancecounter class (system.diagnostics). URL https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.performancecounter. 26

Weijiang Feng, Naiyang Guan, Yuan Li, Xiang Zhang, and Zhigang Luo. Audio visual speech recognition with multimodal recurrent neural networks. pages 681–688, 05 2017. doi: 10.1109/IJCNN.2017.7965918. 13

F.A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: continual prediction with lstm. In *1999 Ninth International Conference on Artificial Neural Networks ICANN 99. (Conf. Publ. No. 470)*, volume 2, pages 850–855 vol.2, 1999. doi: 10.1049/cp:19991218. 35

Guy S. Handelman, Hong Kuan Kok, Ronil V. Chandra, Amir H. Razavi, Shiwei Huang, Mark Brooks, Michael J. Lee, and Hamed Asadi. Peering into the black box of artificial intelligence: Evaluation metrics of machine learning methods. *American Journal of Roentgenology*, 212(1):38–43, 2019. doi: 10.2214/AJR.18.20224. URL https://doi.org/10.2214/AJR.18.20224. PMID: 30332290. 38

Mahmudul Hasan, Md. Milon Islam, Md Ishrak Islam Zarif, and M. M. A. Hashem. Attack and anomaly detection in iot sensors in iot sites using machine learning approaches. *Internet of Things*, 7:100059, Sep 2019. ISSN 2542-6605. URL http://www.sciencedirect.com/science/article/pii/S2542660519300241. 18

Hassan Hassan, Abdelazim Negm, Mohamed Zahran, and Oliver Saavedra. Assessment of artificial neural network for bathymetry estimation using high resolution satellite imagery in shallow lakes: Case study el burullus lake. *International Water Technology Journal*, 5, 12 2015. 13

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, Nov 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735. 18

IM0203. Software architecture. URL https://www.ou.nl/-/IM0203_Software-Architecture. 21

Ying Jiang, Na Zhang, and Zheng Ren. Research on intelligent monitoring scheme for microservice application systems. In *2020 International Conference on Intelligent Transportation, Big Data Smart City (ICITBS)*, pages 791–794, 2020. doi: 10.1109/ICITBS49701.2020.00173. 19

Alexandr Krylovskiy, Marco Jahn, and Edoardo Patti. Designing a smart city internet of things platform with microservice architecture. In *2015 3rd International Conference on Future Internet of Things and Cloud*, pages 25–30, 2015. doi: 10.1109/FiCloud.2015.55. 19

Rodrigo Laigner, Yongluan Zhou, Marcos Antonio Vaz Salles, Yijian Liu, and Marcos Kalinowski. Data management in microservices: State of the practice, challenges, and research directions, 2021. URL https://arxiv.org/abs/2103.00170. 20

S. Liu and R. Kuhn. Data loss prevention. *IT Professional*, 12(2):10–13, 2010. ISSN 1941-045X. doi: 10.1109/MITP.2010.52. URL https://doi.org/10.1109/MITP.2010.52. 18

Matti Mantere, Ilkka Uusitalo, Mirko Sailio, and Sami Noponen. Challenges of machine learning based monitoring for industrial control system networks. In *2012 26th International Conference on Advanced Information Networking and Applications Workshops*, pages 968–972, 2012. doi: 10.1109/WAINA.2012.135. 18

Yair Meidan, Michael Bohadana, Asaf Shabtai, Juan D Guarnizo, Martín Ochoa, Nils O Tippenhauer, and Yuval Elovici. Profiliot: A machine learning approach for iot device identification based on network traffic analysis. *Symposium on Applied Computing*, pages 506–509, 2017a. 18

Yair Meidan, Michael Bohadana, Asaf Shabtai, Martin Ochoa, Nils Ole Tippenhauer, Juan Davis Guarnizo, and Yuval Elovici. Detection of unauthorized iot devices using machine learning techniques, 2017b. 18

M. Munir, S. A. Siddiqui, A. Dengel, and S. Ahmed. Deepant: A deep learning approach for unsupervised anomaly detection in time series. *IEEE Access*, 7:1991–2005, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2018.2886457. URL https://doi.org/10.1109/ACCESS.2018.2886457. 18

Ece Guran Schmidt Murat Soysal. Machine learning algorithms for accurate flow-based network traffic classification: Evaluation and comparison. *Performance Evaluation*, 67 (6):451 – 467, 2010. ISSN 0166-5316. doi: https://doi.org/10.1016/j.peva.2010.01.001. URL http://www.sciencedirect.com/science/article/pii/S0166531610000027. 18

Richard Nagyfi. The differences between artificial and biological neural networks, 2018. URL https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7. 34

Srikanth Namuduri, Barath Narayanan Narayanan, Venkata Salini Priyamvada Davuluru, Lamar Burton, and Shekhar Bhansali. Review—deep learning methods for sensor based predictive maintenance and future perspectives for electrochemical sensors. *Journal of The Electrochemical Society*, 167(3):037552, jan 2020. doi: 10.1149/1945-7111/ab67a8. URL https://doi.org/10.1149/1945-7111/ab67a8. 18

T. T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys Tutorials*, 10(4):56–76, 2008. doi: 10.1109/SURV.2008.080406. 17

Fannia Pacheco, Ernesto Exposito, Mathieu Gineste, Cedric Baudoin, and Jose Aguilar. Towards the deployment of machine learning solutions in network traffic classification: A systematic survey. *IEEE Communications Surveys Tutorials*, 21(2):1988–2014, 2019. doi: 10.1109/COMST.2018.2883147. 19

Duarte Raposo, André Rodrigues, Soraya Sinche, Jorge Sá Silva, and Fernando Boavida. Industrial iot monitoring: Technologies and architecture proposal. *Sensors*, 18(10), 2018. ISSN 1424-8220. doi: 10.3390/s18103568. URL https://www.mdpi.com/1424-8220/18/10/3568. 17

Imran Razzak, Khurram Zafar, Muhammad Imran, and Guandong Xu. Randomized nonlinear one-class support vector machines with bounded loss function to detect of outliers for large scale iot data. *Future Generation Computer Systems*, 112:715–723, Nov 2020. ISSN 0167-739X. URL https://www.sciencedirect.com/science/article/pii/S0167739X19313913. 12

Marta Regis, Paulo Serra, and Edwin R. van den Heuvel. Random autoregressive models: A structured overview. *Econometric Reviews*, 41(2):207–230, 2022. doi: 10.1080/07474938.2021.1899504. URL https://doi.org/10.1080/07474938.2021.1899504. 12

A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, 1959. ISSN 0018-8646. doi: 10.1147/rd.33.0210. URL https://doi.org/10.1147/rd.33.0210. 4

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61: 85–117, Jan 2015. ISSN 0893-6080. URL https://www.sciencedirect.com/science/article/pii/S0893608014002135. 4

Dominique T. Shipmon, Jason M. Gurevitch, Paolo M. Piselli, and Stephen T. Edwards. Time series anomaly detection; detection of anomalous drops with limited features and sparse examples in noisy highly periodic data, 2017. 18

Mayank Swarnkar and Neminath Hubballi. Ocpad: One class naive bayes classifier for payload based anomaly detection. *Expert Systems with Applications*, 64:330–339, Dec 2016. ISSN 0957-4174. URL https://www.sciencedirect.com/science/article/pii/S0957417416303839. 11

Johannes Thönes. Microservices. *IEEE Software*, 32(1):116–116, 2015. doi: 10.1109/MS.2015.11. 19

Alfonso Valdes and Steven Cheung. Communication pattern anomaly detection in process control systems. In *2009 IEEE Conference on Technologies for Homeland Security*, pages 22–29, 2009. doi: 10.1109/THS.2009.5168010. 17

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet a generative model for raw audio, 2016. URL https://ui.adsabs.harvard.edu/abs/2016arXiv160903499V. 18

Steve whims. Windows management instrumentation - win32 apps. URL https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page. 15

Rüdiger Wirth. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, pages 29–39, 2000. 22

Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014. doi: 10.1109/TII.2014.2300753. 19

# APPENDIX FIGURES

```
import sys
import datetime
import pandas as pd
import numpy as np
from pandas.plotting import table
import pyarrow as pa
import pyarrow.parquet as pq
import glob
import requests
import json

class DataCollector:

    splitvalue
    filetime = datetime.datetime.now()

    def __init__(self):
        self.splitvalue = 100
        self.filetime = datetime.datetime.now()

    def __init__(self, pagecount):
        self.splitvalue = pagecount
        self.filetime = datetime.datetime.now()
```

Figure 1: DataCollector includes and global variables

```
def get_filedata(csv_file):
    try:
        page = pd.read_csv(csv_file, sep=';', parse_dates=['Time'], index_col=['Time'])
        self.filetime = min(page.index)
        data = []
        for timeblock in np.array_split(page, self.splitvalue):
            starttime = min(timeblock.index)
            endtime = max(timeblock.index)
            data = pd.concat([timeblock, get_bad_data(starttime, endtime),get_mem_data(starttime, endtime)], axis=1).reindex(timeblock.index)
            data.append(data)
        return data
    except:
        print(sys.exc_info()[0])
```

Figure 2: DataCollector get filedata function pseudo code

```
    def get_bad_data(start_time, end_time):
        bdata = []
        url =f'<historianserver>/ip21DataCollector/api/BadData/{start_time.strftime("%d%m%Y%H%M%S")}/{end_time.strftime("%d%m%Y%H%M%S")}/1'
            .replace(':','%3A').replace(' ','%20')
        headers = {"Accept": "application/json"}
        response = requests.get(f'http://{url}',auth=('*******','******')))
        jDat =json.loads(response.json(), object_hook=date_hook)
        data = pd.DataFrame.from_records(jDat,columns=['Time','BadCount']).set_index('Time')
        return data
```

Figure 3: DataCollector get bad data function pseudo code

```
SELECT ts,round(sum(ng)) as bad
FROM aggregates
WHERE
    ts between TIMESTAMP'yyyy-MM-dd HH:mm:ss'-period*10 AND TIMESTAMP'yyyy-MM-dd HH:mm:ss'
AND
    period = $period
group by ts
order by ts
```

Figure 4: webservice bad data sql query

```
select
    a.ts,
    round(a.rng) as adlgp,
    round(b.rng) as cimmem,
    round(c.rng) as virtmem,
    round(d.rng) as physmem
    from (
    select
    ts,rng
    FROM aggregates
    WHERE
    name ='ADLGP_MEM.PV' and ts between TIMESTAMP'yyyy-MM-dd HH:mm:ss'-period*10 AND TIMESTAMP'yyyy-MM-dd HH:mm:ss'

    AND period = $period) a
    left join (
    select
    ts,rng
    FROM aggregates
    WHERE
    name ='CIMMEM.PV' and ts between TIMESTAMP'yyyy-MM-dd HH:mm:ss'-period*10 AND TIMESTAMP'yyyy-MM-dd HH:mm:ss'

    AND period = $period) b
     on a.ts=b.ts
    left join (
    select
    ts,rng
    FROM aggregates
    WHERE
    name ='VIRTMEM.PV' and ts between TIMESTAMP'yyyy-MM-dd HH:mm:ss'-period*10 AND TIMESTAMP'yyyy-MM-dd HH:mm:ss'

    AND period = $period) c
     on a.ts=c.ts
    left join (
    select
    ts,rng
    FROM aggregates
    WHERE
    name ='PHYSMEM.PV' and ts between TIMESTAMP'yyyy-MM-dd HH:mm:ss'-period*10 AND TIMESTAMP'yyyy-MM-dd HH:mm:ss'

    AND period = $period) d
     on a.ts=d.ts
order by a.ts
```

Figure 5: webservice memory data sql query

```
    def date_hook(json_dict):the
        for (key, value) in json_dict.items():
            try:
                json_dict[key] = datetime.datetime.strptime(value, "%Y-%m-%dT%H:%M:%S")
            except:
                pass # do nothing if the value is not a datetime
        return json_dict
```

Figure 6: datetime converter function

```
def collect_data():
    files = glob.glob('<networkdata file location>\\*.csv')
    for file in files:
        print(file)
        try:
            filedata = get_filedata(file)
            dframe = pd.concat(filedata)
            timetable = pa.Table.from_pandas(dframe, preserve_index=True)
            pq.write_table(timetable, f'<working folder>/data{self.filetime.month}-{self.filetime.day}.parquet')
        except:
            print(sys.exc_info()[0])
    return 1
```

Figure 7: collect function that writes to parquet files

```
import pandas as pd
import numpy as np
from sklearn import preprocessing as prep
import pyarrow.parquet as pq
import glob
from statsmodels.tsa import stattools
from statsmodels.tsa.api import VAR
from sklearn import decomposition

class normalizer:
```

Figure 8: normalizer class definition

```
def no_normalize():
    files = glob.glob(r'<presorted>\*.parquet')
    return pd.concat([pd.read_parquet(file) for file in files])

def normalize():
    files = glob.glob(r'<presorted>\*.parquet')

    df = pd.concat([pd.read_parquet(file) for file in files])
    x = df.values  # returns a numpy array
    min_max_scaler = prep.MinMaxScaler()
    x_scaled = min_max_scaler.fit_transform(x)
    df2 = pd.DataFrame(x_scaled, columns=df.columns)
    return df2

def remove_outliers_and_normalize():
    files = glob.glob(r'<presorted>\*.parquet')

    df = pd.concat([pd.read_parquet(file) for file in files])
    df = df.mask(df.sub(df.mean()).div(df.std()).abs().gt(3))
    df = df.interpolate(method="linear")
    x = df.values  # returns a numpy array
    min_max_scaler = prep.MinMaxScaler()
    x_scaled = min_max_scaler.fit_transform(x)
    df2 = pd.DataFrame(x_scaled, columns=df.columns)
    return df2
```

Figure 9: data normalization function

```
def GetGranger():
    df = remove_outliers_and_normalize()
    x =df[['BytesReceivedDcs','BytesSentIp21']]
    granger =stattools.grangercausalitytests(x, 20)
    return granger

def getVar():

    df = remove_outliers_and_normalize()
    model = VAR(df[['BytesSentIp21','adlgp']])
    for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]:
        result = model.fit(i)
        try:
            print('Lag Order =', i)
            print('AIC : ', result.aic)
            print('BIC : ', result.bic)
            print('FPE : ', result.fpe)
            print('HQIC: ', result.hqic, '\n')
        except:
            continue


def try_pca():
    df = remove_outliers_and_normalize()
    pca = decomposition.PCA(n_components=0.95)
    b = pca.fit_transform(df)
    return b
```

Figure 10: data tests

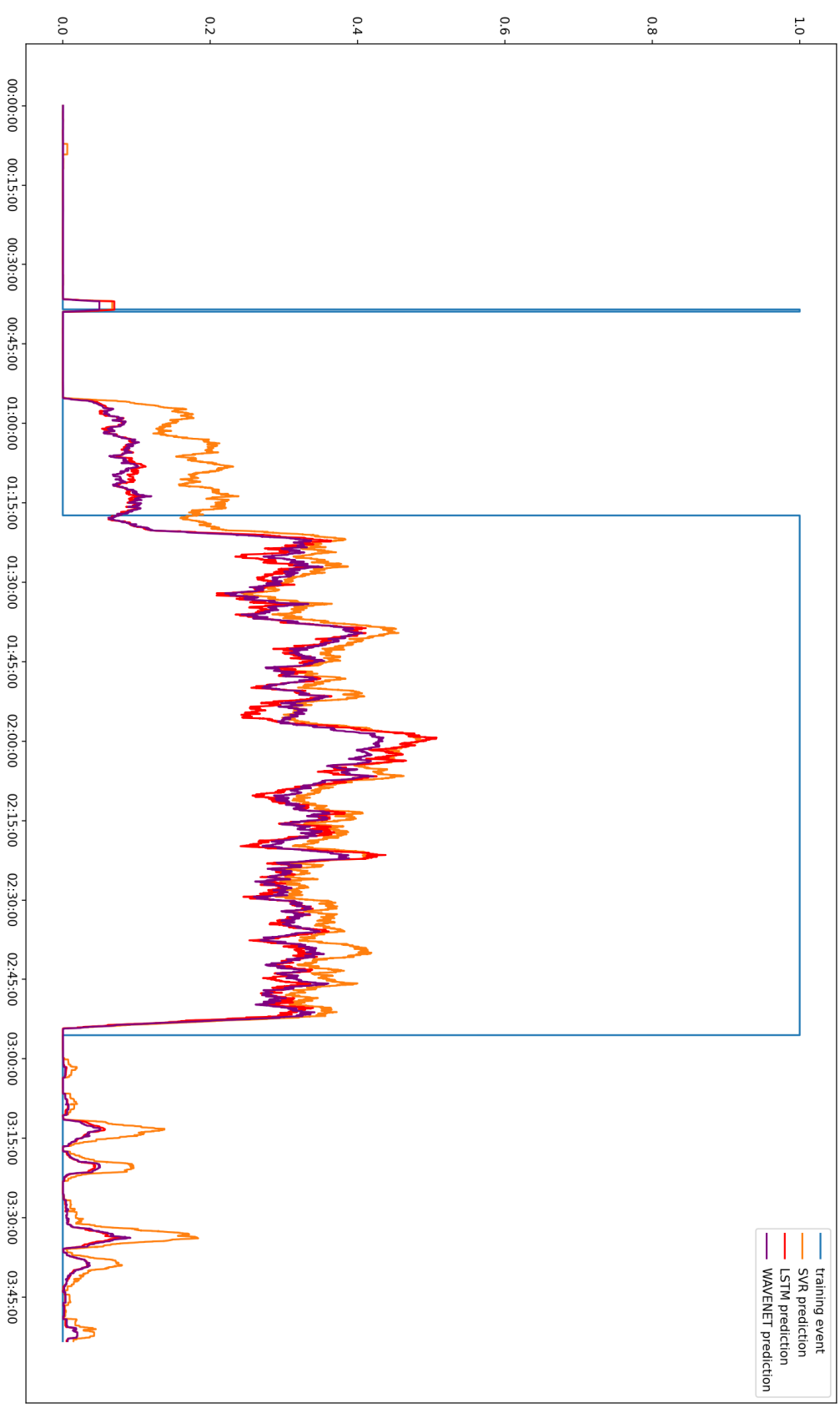


Figure 11: comparison of model predictions for test event