

MASTER THESIS  
ARTIFICIAL INTELLIGENCE  
Radboud University Nijmegen

# The signs are there, now predict the future!

Predicting system failure  
and reliability

ROBERT-JAN DRENTH

[rj.drenth@student.ru.nl](mailto:rj.drenth@student.ru.nl)

s0815357

*Internal supervisor:*

LOUIS VUURPIJL

*External supervisor:*

PAUL VAN DER RIET

*Second assessor:*

MAURITS KAPTEIN

September 12, 2014



## Abstract

All kinds of machines and electronics are finding their way into our everyday lives and we increasingly rely on their proper functioning. Especially when such machines are key to business operations, their reliability is vital. For parking garages this is also the case, where a non-functional machine can lead to both a short and long term loss of income. In order to guarantee that machines stay operational, a generic predictive setup was developed. The basis of this setup was formed by sequences of recorded events, which describe what happens to a device, including regular occurrences as well as system failures.

Such a sequence of events are also known as time-series and our setup is aimed to predict future time-steps of these series, which describe whether each possible event will occur in the future. Such predictions can aid the process of deciding whether proactive maintenance should be performed or not.

For our setup, we tested the Multilayer Perceptron and Conditional Restricted Boltzmann Machine algorithms, as well a Recursive and Single Step paradigm for time-series prediction, which determine how multiple future time-steps are predicted. We furthermore investigated the influence of different data preprocessing methods on the setup's performance, such as the addition of extra features to the dataset, which are generated using Principal Component Analysis, and the application of an outlier treatment method, which enforces a maximum value during the normalisation process.

The two mentioned data preprocessing methods were proven to be ineffective, while we demonstrated that a setup utilising the Conditional Restricted Boltzmann Algorithm, in combination with the Single Step prediction paradigm, was able to create generalised prediction models for devices of different brands. Such a setup achieved similar average accuracy scores of almost 75% for the events within predicted time-steps of the time-series, regardless of a device's type or brand, which is sufficient for its intended real-world application.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Project Context . . . . .	2
1.2	Challenges . . . . .	2
1.3	Possibilities and related work . . . . .	3
1.4	Research Questions . . . . .	4
<b>2</b>	<b>Approach: Test Environment</b>	<b>5</b>
2.1	The pipeline . . . . .	5
2.2	Creating and executing experiments . . . . .	8
2.3	Analysing results . . . . .	8
2.4	Achieving exchangeability of methods . . . . .	9
<b>3</b>	<b>Methods</b>	<b>11</b>
3.1	Preprocessing . . . . .	11
3.1.1	Preparing raw data . . . . .	11
3.1.2	Partitioning a continuous stream into discrete time periods . . . . .	14
3.1.3	Feature Selection . . . . .	16
3.1.4	Regular preprocessing steps . . . . .	16
3.1.5	Generating time series . . . . .	17
3.2	Machine Learning Algorithms . . . . .	18
3.2.1	Multilayer Perceptron . . . . .	19
3.2.2	Conditional Restricted Boltzmann Machine . . . . .	20
3.3	Prediction paradigms . . . . .	22
3.3.1	Single Step paradigm . . . . .	22
3.3.2	Recursive paradigm . . . . .	22
<b>4</b>	<b>Experimental Setup &amp; Results</b>	<b>23</b>
4.1	Experimental Setup . . . . .	23
4.1.1	Experimental parameters . . . . .	23
4.1.2	Evaluation measures . . . . .	25
4.2	Results . . . . .	27
4.3	Discussion of results . . . . .	32

<b>5</b>	<b>Discussion</b>	<b>35</b>
5.1	Assessing generic event prediction capabilities . . . . .	35
5.2	Future research . . . . .	36
5.3	Conclusion . . . . .	37
<b>A</b>	<b>Test environment interface images</b>	<b>43</b>
<b>B</b>	<b>Experiment results: tables reporting MSE values</b>	<b>47</b>
B.1	Baseline experiment . . . . .	47
B.2	Number of historic time-steps . . . . .	48
B.3	Number of predicted time-steps . . . . .	48
B.4	PCA features . . . . .	48
B.5	Cap z-scores . . . . .	49
B.6	Learning rate . . . . .	49
B.7	Epochs . . . . .	49
B.8	Number of hidden units . . . . .	50
<b>C</b>	<b>Experiment results: graphs of evaluation measures</b>	<b>51</b>
C.1	Baseline experiment . . . . .	51
C.2	Number of historic time-steps . . . . .	52
C.2.1	2 historic time-steps . . . . .	52
C.2.2	3 historic time-steps . . . . .	52
C.2.3	5 historic time-steps . . . . .	53
C.2.4	6 historic time-steps . . . . .	53
C.3	Number of predicted time-steps . . . . .	54
C.3.1	3 predicted time-steps . . . . .	54
C.3.2	4 predicted time-steps . . . . .	54
C.4	PCA features . . . . .	55
C.4.1	PCA features covering 75% variance . . . . .	55
C.4.2	PCA features covering 0.95% variance . . . . .	55
C.5	Cap z-scores . . . . .	56
C.5.1	Capping z-scores at 1.75 . . . . .	56
C.5.2	Capping z-scores at 2.00 . . . . .	56
C.5.3	Capping z-scores at 2.25 . . . . .	57
C.5.4	Capping z-scores at 2.50 . . . . .	57
C.6	Learning rate . . . . .	58
C.6.1	Learning rate 0.01 . . . . .	58
C.6.2	Learning rate 0.05 . . . . .	58
C.7	Epochs . . . . .	59
C.7.1	500 epochs . . . . .	59
C.7.2	2,500 epochs . . . . .	59
C.7.3	5,000 epochs . . . . .	60
C.7.4	10,000 epochs . . . . .	60
C.8	Number of hidden units . . . . .	61
C.8.1	0.33 hidden unit ratio . . . . .	61
C.8.2	0.66 hidden unit ratio . . . . .	61

# Chapter 1

## Introduction

Being able to accurately predict the future has obvious advantages for about any ongoing process imaginable. Two applications that will probably come to mind quickly, are weather forecasting [8, 13, 18] and predicting prices on the stock market [3, 14, 17, 28], as knowing when to dress for rain or how to get rich quickly would be very valuable information indeed. An application that appeals less to the imagination, but for which it is also recognised that predicting the future certainly would be valuable, is forecasting the behaviour of machines. Or, to be more specific, predicting when they are about to fail [11, 16, 32]. Considering the fact that we encounter more machines in our daily lives with each passing day, the task of making sure they perform reliably without interruption becomes increasingly important.

Of course, machines cannot run indefinitely without receiving maintenance periodically. Without that, they would eventually break down, causing them to require repairs. Especially for businesses that depend on the machines they employ, such breakdowns can interrupt their business process and potentially cost a lot of money. Next to that, such events could also cause customer dissatisfaction in the situation that customers have to deal with such machines directly and encounter one that is broken. Striving to maintain smooth operations, the practice of performing preventive maintenance is a common occurrence, which aims to prevent breakdowns due to ‘wear and tear’ by performing regular maintenance. However, it is not unimaginable that such maintenance might be done when it is not yet required, or that it turned out to be scheduled too late. This could occur in situations that a machine was subjected to an unexpectedly decreased or increased strain for a certain period of time. Furthermore, unforeseen external events could also influence the performance of a machine. This means that on one side excessive maintenance can cost a lot of money, while on the other side a lack of maintenance can lead to an interrupted business process which can cost even more money, both on a short term and in the long run. Therefore, being able to predict when preventive maintenance actually is required can be a very valuable asset.

The scenarios that were sketched above also hold for the parking garage industry, of which the machines within such a setting are the focus of this study; it is our aim to find a way to predict what is going to happen the machines that are used in parking garages, thereby aiding managers in deciding when to take proactive steps to prevent problems from occurring.

Presently more details regarding the context of this study will be given, followed by an overview of related work and, finally, the research questions that will help us define the project and its focus will be presented.

## 1.1 Project Context

As mentioned briefly in the previous section, the focus of this study is to predict what will happen to the different types of machines that are used in parking garages. Types of present machines include entry barriers, payment machines and ticket machines. For each type of machine there are three different manufacturers for which data is available to us and each manufacturer produces their own version. An operator of a parking garage then selects the brand of machines with which his or her parking garage will be equipped.

Irrespective of the manufacturer that produced a device, they all generate events for everything that happens to them. Most of these are ordinary events, indicating when a customer inserts a ticket or when a barrier opens or closes. Then there are events that could indicate future problems, such as events indicating a low level of change or a low level of blank tickets. Finally, there are events that get generated in case of problematic situations, such as a reader not being able to read a ticket, a barrier not opening or closing, or an event indicating that all change or blank tickets are depleted.

All such generated events that inform the external world of what is happening are sent in a message to a general server for logging the moment they are generated. This means that messages describing the events come in one by one and are stored the moment they are received. Apart from an event number, which identifies the nature of the event, such messages also contain the IP address from where the message originated, a time-stamp, the number of the device that generated the event and, if available, a description of the event.

However, there is no industry standard for the information contained within each message, nor for the possible events that can be generated by a device; both are different across brands. Examples of raw messages, along with extensive descriptions of their properties, are given for each brand in Chapter 3.1.1. A comparison of the number of possible events that the devices of each brand can generate, demonstrates the lack of a standard most prominently; these numbers are 2112, 857 and 107 respectively. This means that no information can be gained with regards to devices of brand A by modeling those of brand B or vice versa, especially in combination with the expected hardware and software differences between machines of different brands.

For this study the data of over 250 parking garages is available, each having a number of machines ranging from 5 to 25. This data describes a continuous period of  $5\frac{1}{2}$  months: from September 29th, 2013 to March 13th, 2014. During this period all messages originating from these parking garages are logged. In the current situation, all messages are simply logged without any further processing taking place. This means that no information is gained from them; they are simply logged for the sake of logging. It is our intention to change this and extract what information can be extracted from these events.

## 1.2 Challenges

In our effort to extract information from all reported events, with the goal of predicting when they will reoccur in the future, we encountered a number of challenges.

The first challenge is the before-mentioned fact that there are three different brands, which can conceptually be viewed as an increase of the number of different types of machines by a factor of three. This is because devices that have a similar function, but are created by different manufacturers, will generate different sets of events and differ on both a hardware and software level.



Secondly, the structure of the messages that report events is different for each brand. However, this is to be expected. But apart from that, they do not all report the same information within such messages either, e.g. brands A and B include one type of information and omit another, while brand C does it the other way around. This means that some form of unification of the incoming information must take place before the data can be further processed.

Third, a list of possible events is available, but unfortunately it is incomplete and a large number of event descriptions are missing. This keeps us from attaching meaning to the majority of events. Furthermore, no list indicating the type of each device was available and, regardless of brand, no indicators of the device type are present within the messages that each device generates. These two facts severely limit the possibilities of incorporating domain knowledge and the ability to generalise devices of the same type.

Finally, the previous point signifies that we must keep track of and predict all events, instead of limiting our predictions to the events that are indicative of a problem. This means our data has a high dimensionality, which will make trained models more complex and restricts the types of algorithms we can use.

### 1.3 Possibilities and related work

Work related to the prediction of machine failures[11], the creation of prognostic tools[16] and planning of maintenance schedules based on estimations of machine degradation [32] has been performed repeatedly in the past. More focused efforts to predict when specific events will occur have also been made. Weiss and Hirsh [29] attempted to predict rare events in event sequences and Sahoo et al. [21] aimed to predict critical events in computer clusters. More recently a survey was conducted to assess failure prediction methods [22] and Wu et al. [31] and Rudin et al. [20] attempted to improve power grid reliability by predicting which parts of the grid were likely to fail.

However, either these papers focused on a too-narrow number of targets that were to be predicted, rendering their methods unusable for our high-dimensional problem [21, 29], or they employed domain knowledge[20, 31], which we are unable to do. Furthermore, since our approach encompasses the prediction of all events, and thereby the prediction of the complete event time-series, the field of time-series forecasting might offer more relevant related work.

Within this area, plenty of research has been performed; Box et al. [6] wrote a book about the subject and various different methods have been surveyed by Ahmed et al. [4]. In the survey the authors compared various types of neural networks and regression algorithms and concluded that, of all evaluated algorithms, the Multilayer Perceptron (MLP) algorithm performed best. This is a well-known and established algorithm and was first introduced to a broad audience by McClelland et al. [19].

Apart from the surveyed algorithms by Ahmed et al. [4], work has also been performed to extend Restricted Boltzmann Machines (RBM)[25] in an effort to develop a method for applying them to the prediction of high-dimensional time series. As a result a new algorithm was developed; the Conditional Restricted Boltzmann Machine (CRBM) [27]. This algorithm was successfully applied to model pigeon behaviour [33] and human motion [26], thereby demonstrating its success in predicting high-dimensional time series.

Together, the established MLP algorithm and the new, but promising state-of-the-art CRBM algorithm, provide a good balance and basis for this work. They will be applied and their performance will be investigated.

Apart from considering different algorithms to achieve the goal of forecasting time series, the effects of different prediction paradigms have been evaluated as well. In the work of Bontempi et al. [5] two distinct categories of prediction paradigms are defined: the *Single Step* and *Recursive* paradigms. A Single Step paradigm predicts all of the required data, which can consist out of multiple time-steps, with a single prediction, while a recursive paradigm requires multiple iterations of predicting single time-steps and builds on previously predicted time-steps. Which paradigm is suitable can vary, depending on the application. Therefore, both paradigms will be evaluated in this project.

It can be observed that a lot of related research has been performed in the fields of machine failure prediction [11, 16, 32], the prediction of specific events [20, 21, 22, 29, 31] and time-series forecasting [4, 6, 26, 33]. The variety of approaches used in the literature show that every problem has its own solution and that the right combination of methods has to be found from all available options for this project.

## 1.4 Research Questions

From the previous chapter on the challenges of this project, Chapter 1.2, it is clear that there are certain problems that must be tackled in order to complete this project successfully.

The main challenge is the diversity of the data that we are dealing with, since there are three different brands, which each has its own large sets of events that their devices can generate. In order to prevent having to reinvent the wheel by designing a new system for each combination of brand and device, we must try to create a generic setup that will be able to make reliable predictions in every situation, regardless of brand or device.

Secondly, considering we do not want to reinvent the wheel, we must investigate whether existing methods or algorithms might be suitable in this situation or if something novel must be designed.

These two goals describe what we intend to achieve with this work and they can be translated into the following research questions:

1. Are existing methods and algorithms that are utilised in related fields applicable for the task of predicting the events generated by the systems present in parking garages?
2. Is it possible to create a single generic, predictive model that can be used for the different system types?

In order to answer these questions a test environment was developed to aid with the process of creating and customising an experimental pipeline. This test environment will be discussed in Chapter 2, while the different considered preprocessing methods, machine learning algorithms and prediction paradigms will be discussed in Chapter 3. Chapter 4 will present the performed experiments and their results, accompanied with an evaluation. This will be followed by a general discussion and conclusion of the complete project in Chapter 5.

# Chapter 2

## Approach: Test Environment

We approach the problem of predicting the event time-series, as summarised in Chapter 1.4, by first creating the right set of tools that it requires. Specifically this means the creation of a pipeline that exhibits a certain versatility, by enabling us to easily add or exchange different parts of the overall process, without giving rise to compatibility issues. Exchangeable parts should include the different steps of the preparation and preprocessing of data to the training of models and the analyses of their performance.

With this goal in mind, a test environment was created that was inspired by the data mining environment WEKA [10], which is capable of performing similar tasks to what is required for this project. However, while WEKA has a convenient interface that makes it easy to use, its internal structure is complex and has a rather steep learning curve. This makes the task of extending the existing environment with new algorithms difficult and it adds a lot of overhead to make new functionality fit. When it was attempted to incorporate the jaRBM library (see Chapter 3.2.2), it proved to be more difficult to do so than to create a new custom test environment.

This new test environment is responsible for:

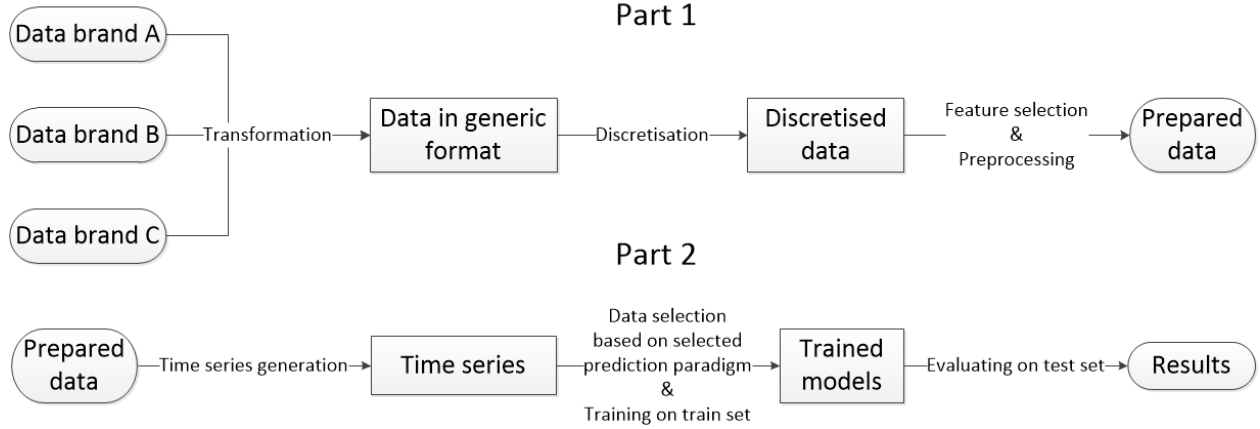
- The creation and tuning of experimental pipelines
- Executing the experiments
- Providing the tools for analysing the results of the experiments

In this case, an experiment is a complete pipeline with its own unique combination of parameters.

How it achieves these three responsibilities, along with how it enables the exchanging of different methods and algorithms without compatibility problems, will be explained in this chapter.

### 2.1 The pipeline

The pipeline within the test environment takes care of the complete process of creating prediction models from start to end. This process consists of many different steps, which will be briefly mentioned here and detailed in Chapters 3.1 through 3.3, and is visualised in Figure 2.1. These steps can be assigned to two categories, which take shape as different parts within the process; the preparation of raw data and the creation and evaluation of prediction models.



**Figure 2.1:** Visualisation of the pipeline of an experiment as created by the test environment. The different steps are further explained in Chapter 3 and are formally defined in Equations 2.1 - 2.4.

The first part of the process consists of various steps to prepare the available data so that its final representation allows machine learning algorithms to get the most out of it. The first step of this part is the transformation of the different data formats of each brand to a common format. After that has been achieved, the continuous stream of data must be discretised into time-steps of a predefined duration. Each time-step is described by a vector of numbers, indicating the amount of times each possible event has occurred within that period. This vector is the feature vector in its most basic form and is the basis for more general methods of preprocessing, which are applied in the next steps. Of the preprocessing methods, feature selection is applied first to reduce the number of features. After this has been done, further optional preprocessing steps can be performed, such as standardisation or normalisation. Finally, time series are created, which in turn will serve as input for algorithms, thereby concluding data preparation.

The second part of the process is the training of machine learning algorithms on the prepared time series and evaluating them. At this stage, models are sequentially generated for each device present within each parking garage, in an attempt to model their behaviour. For each device, the data that describes its past behaviour is split up in multiple train and test sets using 5-fold cross-validation<sup>1</sup>, upon which the models are trained and tested. The results of these tests, which are the values predicted by each model, are then saved for future analysis.

Finally, the pipeline undertakes the necessary steps in order to support different prediction paradigms (which paradigms are supported and what they encompass will be explained in Chapter 3.3). It achieves this with an intermediate step that accepts time series data and controls what data reaches the prediction algorithms, possibly altering it in the process by incorporation the results of previous predictions, depending on the selected prediction paradigm.

<sup>1</sup>5-fold cross-validation was used, as the amount of available data, in combination with the selected duration for each time-step and the number of time-steps used for generating the time series, resulted in a very low amount of data entries. This required us to choose a lower amount of folds than the generally accepted number of 10 folds.

The general steps that the pipeline performs, from the transformation of raw data to the generic format, to the prediction of future time-steps, can be expressed in mathematical formulas, thereby formally defining the complete process. These equations can be found below. Equation 2.1 expresses the transformation of the data of individual brands to the generic representation, Equation 2.2 the discretisation and counting of events, Equation 2.3 the preprocessing of the data and Equation 2.4 the prediction of new data by the generated models.

In the final equation, Equation 2.4, the resulting predicted values  $e''$  are from a larger domain than the preprocessed values  $e'$  as stated in Equation 2.3. The actual predicted values depend on the used machine learning algorithms. How these predictions are interpreted is discussed in Chapter 4.1.2.

### Transforming data to generic format:

$$y = \sum_{i=1}^n T_i(x_i) \quad (2.1)$$

where:

$y$  = sequence of events in the generic format

$n$  = total number of brands

$x_i$  = sequence of events for brand  $i$

$T_i(x_i)$  = function that transforms the data of brand  $i$  to the generic representation

### Discretising the data:

$$e = f(y) \quad (2.2)$$

where:

$e$  = discretised event counter vectors with all  $e_i \in \mathbb{N}$

$f(y)$  = function that discretises the events and counts the occurrence of each event within every time-step

### Feature selection & preprocessing:

$$e' = p(e) \quad (2.3)$$

where:

$e'$  = preprocessed event counter vectors with all  $e'_i \in [0, 1]$

$p(e)$  = all preprocessing steps together

### Time-series prediction:

$$p(M_n, e'_{[t-h,t]}, k) = e''_{[t+1,t+k]} \quad (2.4)$$

where:

$M$  = prediction model that is trained to predict  $n$  time-steps at a time

$e'_{[t-h,t]}$  =  $h$  consecutive historic time-steps preceding and including time-step  $t$

$k$  = number of time-steps that is to be predicted

$p$  = prediction paradigm that uses model  $M$  and time-steps  $e'_{[t-h,t]}$  to predict  $k$  subsequent time-steps.

$e''_{[t+1,t+k]}$  =  $k$  predicted event counter vectors (time-steps) subsequent to time-step  $t$ , with all  $e''_i \in \mathbb{R}$

## 2.2 Creating and executing experiments

The actual creation of a pipeline and the fine-tuning of its parameters is split up in different parts of the test environment’s interface. All parameters of the pipeline are split up in the following categories:

- **Preprocessing:** Allows various settings with regards to the preparation of raw data and the supported preprocessing steps.
- **Time series generation:** Contains setting with regards to how many time steps are used for predictions and how many steps are to be predicted.
- **Prediction model and training parameters:** Includes parameters with regards to cross-validation, the selected machine learning algorithm and the type of prediction paradigm that is to be used.

After setting the parameters for each category, a textual representation of the settings is displayed in an overview of the experiment. Images of these parts of the interface, including an image showing the experimental overview from which they are accessible, are shown in Figures A.2, A.3, A.1 and A.4 respectively in Appendix A.

Once all the parameters of an experiment have been set, the experiment can be saved to the hard drive, allowing it to be retrieved from memory later, after which it can be altered and be re-saved, possibly under a different name. This enables us to easily create a large number of different experiments, varying a single parameter each time, in order to determine the influence of each parameter on the model’s performance. The test environment conveniently supports loading multiple experiments into memory at the same time, thereby scheduling them for execution, which can be done by simply clicking the button assigned for this task. It will then execute a predefined number of experiments in parallel by utilising multiple computational threads. In order to further increase efficiency, it allows experiments to reuse either prepared data or preprocessed time series, so that such steps are not performed redundantly, thereby saving a lot of time for each experiment.

## 2.3 Analysing results

The test environment also provides the tools to further analyse the results of each tested model. It was briefly mentioned that the pipeline saves the predictions that are made by each model. Along with those predictions the true values of the data are also saved. Saving both allows the environment to calculate various different measures, such as the True positive rate or Accuracy, which can be used for the evaluation of the results. The applied measures, what information they yield and how they are calculated, will be explained in detail in Chapter 4.1.2.

Since the predicted values are continuous in nature, we can vary the threshold at which point a prediction is considered to be indicative of the occurrence of an event. Varying this threshold also influences some of the measures that we use to evaluate the performance of a model, which in turn allows us to plot the relation between different threshold levels and the used measures. The test environment facilitates the creation of such plots and they will be extensively used in Chapter 4.2 for the comparison of results.

## 2.4 Achieving exchangeability of methods

In order to meet the requirement of versatility, by being able to exchange one preprocessing method or an algorithm for another, certain steps had to be taken. A closer look at WEKA revealed that a system which explicitly states the required input and the resulting output formats of data, for both preprocessing methods as well as machine learning algorithms, is able to achieve this goal. Enforcing such restrictions makes sure that results are what you expect them to be and that incompatible combinations of methods are not possible. We adopted a similar system to meet our needs.

For the different preprocessing methods the requirements were the same for input data and the resulting output, in order to allow for the scheduling of one process after another and guaranteeing exchangeability. This requirement states that the input and result of a method must both be a matrix of objects, of which the contents of each column must be of the same data type. Should a preprocessing method be unable to handle a certain type of data within a column, it is expected to simply skip it.

Of implemented algorithms it is required that they minimally support a number of functions in order to be accepted. The required functionality incorporates the ability to accept a vector or matrix of decimal numbers and train itself, test itself or make predictions<sup>2</sup> based on the given vector or matrix.

As long as any preprocessing method or machine learning algorithm adheres to these rules, they can be placed in the pipeline without difficulties.

---

<sup>2</sup>The difference between testing and making a prediction is that, when testing an algorithm, its results can be compared to known labels or values and report a certain score, whereas when it has to make a prediction such labels might not necessarily be available.





# Chapter 3

## Methods

This chapter will explain the preprocessing steps of turning raw data into a representation suitable for machine learning in detail. These steps were previously visualised as a part within the complete pipeline in Figure 2.1 and formalised in Equations 2.1 - 2.3. Furthermore, the machine learning algorithms that were selected and compared will be discussed, as well as the considered prediction paradigms.

### 3.1 Preprocessing

For machine learning problems, turning the available raw data into a representation which allows for the extraction of the highest amount of information by machine learning algorithms, is one the most important steps to attain good results. In Chapter 2.1 the pipeline that performs these steps, as well as the order in which they were performed, were discussed. In this chapter we will discuss each required preprocessing step and how the desired result is achieved, in the same order.

#### 3.1.1 Preparing raw data

The raw data that was available to us consisted of messages containing reports of events that occurred to the devices within parking garages. Furthermore, we had to incorporate the data from three different brands, each with their own message format. These three different formats will be discussed first, followed by the method with which we transformed them to a common, generic format, as expressed in Equation 2.1.

##### Raw data format

Each brand has its own format of reporting messages, as there is no industry standard. Because of this, differences between these formats are to be expected. However, despite these possible differences, we can assume that each format contains the same kind of information, be it in their own specific representation, as each brand must be able to report events in some way. Key information that each message should convey at the very least, include:

- An event number
- Originating location
- Originating device

	Brand A	Meaning
1	10.131.208.156	IP Address
2	WPS	Brand name
3	0	Unknown number
4	8.13.04	Multiple event numbers
5	32	Fourth event number
6	11	Device number
7	Deur uit	Event description
8	2013-09-24 17:23:35.000	Receiver timestamp
9	2013-09-24 17:23:49.967	Local timestamp
10	NL/HL/LN	Location description
11	32	Repeating fourth event
12	Deur uit	Repeating event description

**Table 3.1:** Message format for Brand A. Note the presence of an ‘unknown number’ and multiple different event numbers, of which the fourth seems to be the most important one, as it is repeated. Furthermore, there is no field for the originating location.

- Time of occurrence

This is the bare minimum that is required for such messages to be useful and forms the basis for a common format. Anything else that is reported, is extra information. Using the provided documentation on the message formats of each brand and careful examination of the data, we were able to map what kind of information was reported.

Examples of messages for each brand are given in Tables 3.1, 3.2 and 3.3, along with a label of each part of the message. All messages were originally reported in the *comma separated values*(CSV) format, but the examples are displayed in tables for convenience. As a general note, all examples have two timestamps. Within these tables, the first time-stamp indicates the local time of the message’s originating location. The second time-stamp indicates the time when the message was received by the operating center, which is added automatically and was retrieved from the database in which all messages were collected.

## Transforming raw data to generic format

From the tables displaying the different message formats, Tables 3.1, 3.2 and 3.3, it becomes clear that transforming the data to a generic format would be very helpful, due to their differences. However, the format of Brand A does not have a field for the originating location, which is considered a bare minimum in the previous section. In order to solve this problem we used the reported IP addresses as an indicator of the originating location, as they are unique. Furthermore, Brand A’s format reports 4 messages at the same time. This problem got resolved by creating separate entries in the new format for each of these events.

It was also stated that two time-stamps are present for each message. It was decided that the second (receiver) time-stamp should be used, as it provides a consistent indicator of time across the different locations, especially since the reported time and date of individual locations were not always correct.

	Brand B	Meaning
1	10.131.208.149	IP Address
2	SandB	Brand name
3	Event:8712	Text : Garage number
4	8712701	Garage number concatenated with device number
5	15360	Event number
6	2013-09-24 17:27:48.000	Local timestamp
7	2013-09-24 17:23:56.060	Receiver timestamp
8	NL/RY/BO	Location description

**Table 3.2:** Message format for Brand B. This format is the most concise of all brands and contains the bare necessities. Note that in fields 3 and 4 the garage number is repeated and that it is concatenated to the device number in field 4.

	Brand C	Meaning
1	10.131.208.148	IP Address
2	SkiData	Brand name
3	UserEvent:504506	Event type : Garage number
4	14	Device number
5	76	Event number
6	2013-09-24 17:24:04.000	Local timestamp
7	2013-09-24 17:23:57.013	Receiver timestamp
8	NL/AM/MU	Location description
9	14	Repeating device number
10	76	Repeating event number
11	Terminal connection fault	Event description

**Table 3.3:** Message format for Brand C. This format has a straightforward composition and the only aspect worth mentioning, is that it also reports the type of an events in field 3. The different possible types are ‘*DeviceEvent*’, ‘*UserEvent*’ and ‘*SystemEvent*’.

	Generic format
1	IP Address
2	Brand name
3	EventType
4	Receiver timestamp
5	Device number
6	Event number
7	Garage number
8	Location description
9	Event description

**Table 3.4:** Generic message format. Fields 3 and 7 to 9 contain extra information that is not used further down the pipeline, but is included where possible, should it ever be required.

In combination with the data present in each message, these decisions allowed us to create a generic format that contains the bare minimum of required information of events that occur. This new, generic format is displayed in Table 3.4. It contains all minimally required information and, where available, extra descriptive data. When additional descriptive data was not available, the field was simply left blank in the transformation process. This is also the case for field 7, ‘garage number’. Although this number was present for two of the three brands, it is considered to be extra data that might be useful for future applications and informative in manual inspections, but is ignored by our application.

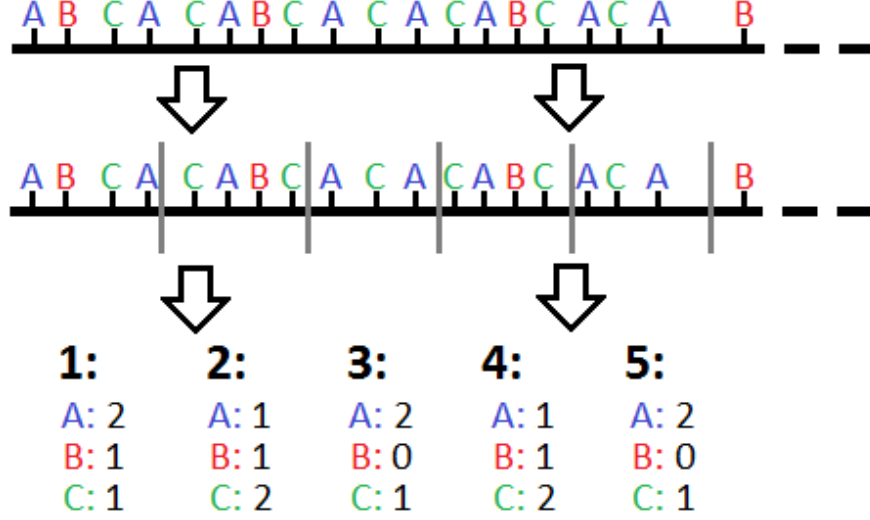
This generic format provides a clearly defined basis, which is the starting point for further processing, making the overall process easier.

## Splitting the data

The next step in the process of preparing the data is relatively small, though significant. After transforming the data into a generic format, all messages are split up based on location and device. This results in separate streams of events for each individual device, which is useful as it ensures that data from different devices does not accidentally get mixed and influences later processing steps. This step does not change the representation of the data, but rather the way it is stored. Previously all the data was stored together in a single file, while after this step data for each device was stored in separate files. Subsequent steps are then performed repeatedly on each individual file, but will be discussed as if they were performed on just one file.

### 3.1.2 Partitioning a continuous stream into discrete time periods

So far the data preparation process transformed the raw data into a generic format, which describes each event with a single line of data. This means that each event is recorded individually and ordered chronologically, so it can be viewed as a continuous stream of data. Such a representation can make it more difficult to find or develop suitable machine learning algorithms that can make predictions. Instead, partitioning the continuous data stream into discrete time-steps could provide a more suitable representation. This part of the overall process is formalised in Equation 2.2.



**Figure 3.1:** Visualisation of the process of transforming a continuous stream of events into a vector representation of event-counters, based on discrete time-steps. At the top a continuous stream is displayed. Between the arrows borders between different time-steps are imposed within the stream, thereby discretising the data. At the bottom, counters of how many times each event occurred within each time-step are displayed. These counters form the vectors of the new data representation.

	Event A	Event B	Event C
Entry 1	2	1	1
Entry 2	1	1	2
Entry 3	2	0	1
Entry 4	1	1	2
Entry 5	2	0	1

**Table 3.5:** Table showing the resulting entries in a dataset, after transforming the example continuous stream of Figure 3.1 into discretised time-steps and counting how many times each event occurred within each time-step.

Creating such a partitioning is achieved by first defining the length of the time-steps. We opted to use time-steps of weeks, as opposed to time-steps of days or months, as it provided a trade-off between being able to make long-term predictions and the number of resulting time-steps that could be created from the amount of data available to us; a period of  $5\frac{1}{2}$  months.

After defining the duration of each period, the number of times each possible event occurs in each period is counted. As a result, the data is now represented in vectors of counters, within which each number is a counter for an event. By doing this we lose the exact order in which events occur within each time-step, but we create a representation that is more suitable for machine learning algorithms, since the counter vector can be used as a vector of features.

A visual representation of this process is displayed in Figure 3.1 and the respective entries in a dataset that the example data, as displayed in the figure, would translate to, is given in Table 3.5.

### 3.1.3 Feature Selection

It was mentioned briefly in Chapter 1.1 that each brand features a large number of possible events: 107, 2112 and 857 for respectively brand A, B and C. Taking into account that we keep track of counters for each event and that we will each of them, we will most likely encounter some dimensionality problems if we would attempt to train models directly on this data.

In an effort to prevent such problems, we can select and keep only features that are meaningful. Since not every event has a known description, we unfortunately cannot apply domain knowledge. However, once more considering the high amount of possible events, it is not unlikely that a large number of those events are very rare, or are limited to certain types of devices. This allows the application of the relatively straightforward technique of removing all counters that never get increased, for each device. This approach was much more successful than anticipated, as the average resulting number of counters was 30.3. This was deemed a small enough number to work with. This step in the overall process of preprocessing the data, along with the steps described in Chapter 3.1.4, is formalised in Equation 2.3.

### 3.1.4 Regular preprocessing steps

While all the previously described steps of preparing the data were tailored to our dataset in order to transform it to a useful representation, the next preprocessing steps that the pipeline allows are more regularly utilised methods.

#### Standardisation

First the technique of standardisation is applied, as it has been shown that Neural Networks perform better on data with properties resulting of this technique [15, 23, 24]. This technique remaps all features so that each has a mean value of 0 and standard deviation of 1. This transforms each individual value  $j$  of feature  $i$  into their *standard score*, also known as the *z-score*, and is calculated using the following formula:

$$z_j = \frac{x_j - \mu_i}{\sigma_i}$$

where  $z$  is the z-score,  $x$  is the original value,  $\mu$  is the mean of the feature and  $\sigma$  the feature's standard deviation. A negative score indicates a value below the mean, while a positive score indicates a value higher than the mean.

Furthermore, this technique can also be applied as a method for treating outliers. Outliers can have a negative impact on the performance of a trained model, as they have proportionally high or low values and can therefore have a much higher impact on results than intended. Within the data, outliers will translate to relatively large z-scores (ignoring the sign of the score), as 95.46% of the z-scores lie within the range of  $[-2, 2]$  for a normal distribution. Using this knowledge, it is possible to use these scores as an indicator to detect outliers. Outliers can then be treated by capping the z-score that will be assigned to them, thereby bringing them more in line with regular scores.

A more common method for treating outliers is to simply remove the entire entry containing the outlier from the dataset. However, considering the little amount of data available to us, as well as the fact that removing entries will result in gaps within our time-steps, we could not apply such techniques.

## Normalisation

The second utilised technique is normalisation. The process of normalisation maps the values of a feature to within a range of  $[0, 1]$ , thereby simply rescaling the data. This can be useful, since some machine learning algorithms perform better with feature values within this range [15, 23]. Calculating a normalised score for a feature is done using the following formula:

$$x' = \frac{x - X_{min}}{X_{max} - X_{min}}$$

where  $x'$  is the new normalised score,  $x$  is the original value and  $X_{min}$  and  $X_{max}$  are respectively the minimum and maximum observed values of a feature.

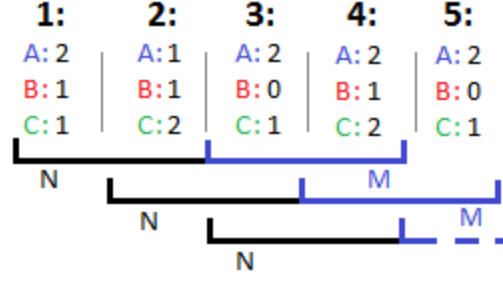
Furthermore, having values within this range is specifically useful for the current application, as normalisation provides clear upper and lower boundaries. While the upper boundary is less important, the lower boundary of zero provides a clear point at which the preprocessed data encodes an event counter value of zero. This is intuitive, but more importantly, it provides a consistent representation of the value zero for all features, which would otherwise all be different if only standardisation was applied. This also means that any predicted value above zero indicates the occurrence of an event. Since we are only interested in the case that an event is going to occur, rather than the exact number of times it is predicted that it will occur, we don't need to transform the predicted values back to the unprocessed form of a counter, due to the clear distinction between the occurrences of an event and the lack thereof.

## Adding Principal Component Analysis features

A final, optional, preprocessing technique that the pipeline supports, is the addition of extra features using Principal Component Analysis (PCA) [1]. It may seem counter-intuitive to add more features after performing feature selection. However, Principal Component Analysis finds an alternative, possibly useful, representation of the data by calculating a new set of features, such that the first features capture the most variability of the data. These are generally considered to describe the underlying structure of the data. Opposed to first features are the last, which capture the least variability and are considered to describe the noise that is present in the data. Therefore, adding a number of the first PCA features to the existing features, might give machine learning models access to the underlying structure of the data without all the noise that is present in the original representation. This makes the addition of extra features generated with PCA worth considering.

### 3.1.5 Generating time series

After the discretisation and preprocessing of the data we have obtained feature vectors for each resulting time-step. However, the current representation of the data does not capture the temporal aspect, other than the order in which each vector appears. A more convenient representation would make the temporal aspect intrinsic to the vectors themselves. This has a couple of benefits, as it allows us to shuffle the data during training (not applied) or randomly pick vectors for cross-validation, instead of just the first or last rows, as the temporal aspect will remain intact.



**Figure 3.2:** Visualisation of the creation of time series from regular time-steps, using 2 time-steps to predict the next 2 steps.

	$A_1$	$B_1$	$C_1$	$A_2$	$B_2$	$C_2$	$A_3$	$B_3$	$C_3$
Entry 1	2	1	1	1	1	2	2	0	1
Entry 2	1	1	2	2	0	1	1	1	2
Entry 3	2	0	1	1	1	2	2	0	1

**Table 3.6:** A new time-series dataset, created from the example data in Table 3.5 by using 2 time-steps to predict 1 time-step. Within the table, columns  $A_1 - C_2$  contain the time-steps that should be used to predict the time-step that is displayed in columns  $A_3 - C_3$ . Entry 1 in this table is created using entries 1-3 from Table 3.5, entry 2 uses entries 2-4 and entry 3 uses entries 3-5. Note that this example does not display the results of any preprocessing steps.

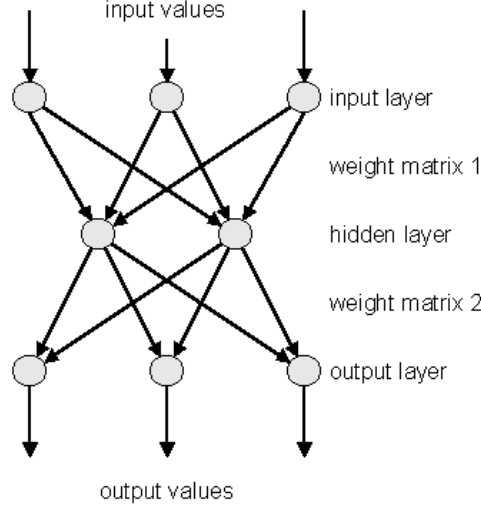
Letting each entry in the dataset capture the temporal aspect is rather straightforward; it can be achieved by creating new feature vectors by concatenating sequential vectors from the discretised dataset. Before doing this, it must first be decided how many steps should be used to create a new vector. This is based on how many time-steps should be predicted and how many steps should be used to make predictions. If, for example, 4 steps will be used to predict the next 2 steps, a total of  $4 + 2 = 6$  steps should be used to create the new vector. This process is visualised in Figure 3.2 and an example dataset, resulting from this process when the previous example data of Table 3.5 is used as input, is given in Table 3.6.

The generation of time-series concludes all the necessary preprocessing steps to transform the raw data into a suitable representation for machine learning algorithms: each row in the data set now is a vector of doubles, which contains both the input and corresponding output.

## 3.2 Machine Learning Algorithms

From the literature two promising algorithms were selected which will be applied and compared. The first algorithm is the Multilayer Perceptron (MLP), which was shown to perform best from a set of compared algorithms on various time series prediction tasks[4]. The second algorithm is the Conditional Restricted Boltzmann Machine (CRBM), which performed well in predicting high-dimensional time-series [26, 27, 33]. These algorithms and how they are utilised was previously expressed in Equation 2.4 and will be explained further in this chapter.





**Figure 3.3:** Visualisation of a Multilayer Perceptron.

### 3.2.1 Multilayer Perceptron

The Multilayer Perceptron and its learning algorithm, backpropagation, have been around for quite some time [7, 30]. They became generally accepted methods after the work of the PDP research group by McClelland et al. [19]. Despite its comparatively long existence, it still is a successful machine learning algorithm.

A MLP network consists out of three or more layers, of which one serves as an input layer and one as the output layer, while the rest of the layers are hidden layers connecting the input and output layers. Each layer consists out of a number of nodes and these nodes are connected to all nodes of the next layer. A visual representation of an MLP network is given in Figure 3.3.

Connections between nodes are called weights and are used to calculate the input to each node. An activation function is then used to calculate the level of a node's activation. Different activation functions with varying mathematical properties can be chosen, depending on the user's needs [15].

Since we normalise the data, which causes its values to lie within a range of  $[0, 1]$ , the standard Sigmoid activation function is suitable for our application, as this function also has an output range of  $[0, 1]$ .

Thus we use the Sigmoid activation function, of which the formula is given below:

$$f(x) = \frac{1}{1 + e^{-x}}$$

When one wishes to calculate the output corresponding to an input feature vector, the values within the vector serve as activation-values for the input nodes. Then, layer by layer, the input and activation values of the nodes in subsequent layers are calculated. The activation values of the nodes in the final output layer serves as the network's output.

Learning is achieved by calculating the difference between a network's output and the desired output, which yields an error value for each node. Using the backpropagation algorithm, these values are used to calculate the error of each individual node of each previous layer, which in turn are used to adjust the weights of incoming connections to a node.

Just as our time-series feature vectors can serve as direct input for the input layer, the activation of the output units can directly be interpreted as predicted time periods.

### 3.2.2 Conditional Restricted Boltzmann Machine

The Conditional Restricted Boltzmann Machine (CRBM) algorithm stems from the original Boltzmann Machine [2], which is also a type of neural network. However, unlike the MLP, it doesn't consist of layers, but rather of a network of fully connected units. Some of these units are observable and act as both input and output units and are called 'visible units'. All other units are unobservable and act as hidden units. Just as is the case for the MLP algorithm, connections between units act as weights with the purpose of calculating the input to each unit. But, in contrast to MLPs, calculating an unit's activation value happens one at a time, due to the network's topology.

However, the training process of basic Boltzmann Machines is considered impractical due to computational issues. This led to the development of Restricted Boltzmann Machines (RBMs) [25], which prohibits connections between visible units and between hidden units, thereby effectively introducing layers just like an MLP network has layers. Training these new RBMs was considered easier, but the computational complexity remained a problem until a fast learning algorithm was invented [12]. Furthermore, RBMs also allow for the use of continuous values, rather than binary values which the original Boltzmann Machines utilised [9].

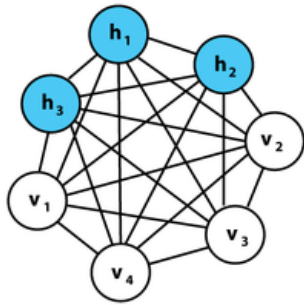
The principals of RBMs are also applicable to the prediction of time-series and to this end the Conditional Restricted Boltzmann Machine (CRBM) was developed, which was shown to be able to predict high-dimensional time-series [26, 33, 27]. It manages this by incorporating the temporal aspect of time-series by copying the state of the visible units to a set of hidden units after every iteration. It thereby conserves information from one iteration so it can be used in the next. These extra hidden units are not updated when all the other units are updated, but rather provide passive, constant input. Should it be desirable that multiple states are remembered, multiple sets of such passive units can be added in which the values of visible units from multiple iterations can be stored.

A comparative visualisation of the three different types of Boltzmann machines (the original Boltzmann Machine, the Restricted Boltzmann Machine and the Conditional Restricted Boltzmann Machine) is displayed in Figure 3.4.

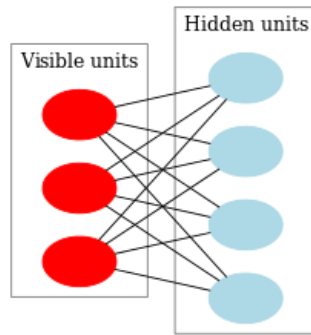
Since continuous values are allowed for these models (as they are extensions of the regular RBMs), our time-series feature vectors can be used to set the activation values of the active visible units. While normally for RBMs the visible units serve both as input and output units, the open-source jaRBM library<sup>1</sup>, which was used for this project, adds the required number of output values as extra visible units to the visible layer. Just like the output units of the MLP, the states of these extra visible units can be interpreted as the output of the model.

---

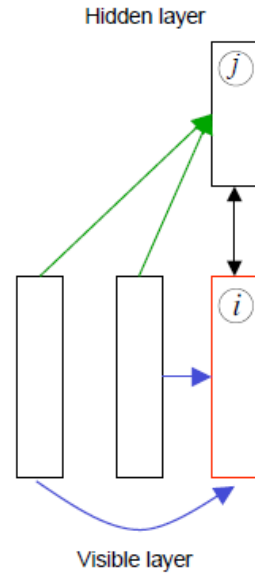
<sup>1</sup>jaRBM: A Java Library for Restricted Boltzmann Machines. <http://sourceforge.net/projects/jarbm/>



(a) A visualisation of the topology of the original Boltzmann Machine. The network is fully connected and the hidden nodes are highlighted.



(b) A visualisation of the topology of a Restricted Boltzmann Machine, emphasising that there are no longer connections between nodes within the same layer.



(c) An abstract visualisation of the topology of a Conditional Restricted Boltzmann Machine, featuring extra units for two previous time-steps. Image taken from [26].

**Figure 3.4:** A visual comparison of the three different types of Boltzmann Machines

### 3.3 Prediction paradigms

When it is required to only predict a single time-step at a time, there is just a single approach that can be taken on a high level; you simply let your model make the prediction. However, when the aim is to predict more than a single time-step, multiple options become available [5]. Two paradigms that will be compared, are the *Single Step* and *recursive* paradigms. The role they play in making predictions is expressed in Equation 2.4 and will be explained further below.

#### 3.3.1 Single Step paradigm

The Single Step paradigm predicts multiple time-steps in one single prediction iteration. This is achieved by training a machine learning algorithm to give all the required predictions at the same time. This requires the algorithms to create more complex models in order to capture the information required to make such predictions. While having complex models is not necessarily negative, it will at the very least have computational consequences, but it also increases the chances of over-fitting.

With regards to Equation 2.4 it must be noted that this paradigm requires a prediction model trained to predict  $k$  time-steps. The result of this, is that the  $n$  parameter in the equation is constrained to  $n = k$  when this paradigm is selected.

#### 3.3.2 Recursive paradigm

The recursive paradigm makes its predictions, as the name implies, by applying recursion. It predicts multiple time-steps by repeatedly predicting single time-steps. The first time step it predicts, is based on the  $N$  historic time-steps that serve as input for the model. After the first prediction has been made, it is treated as if it were real, observed data and the next prediction is made using the historic time-steps except for the oldest one, and treats the previously predicted time-step as the most recently observed data. This is repeated until the required amount of time-steps has been predicted. In respect to Equation 2.4, this means that a prediction model is required for which  $n = 1$ .

While the utilisation of this paradigm results in models of a lesser complexity compared to the models created using the Single Step paradigm, it has the drawback of an accumulation of errors. This is caused by the fact that, as predictions are made further into the future, we rely more and more on previously estimated values, which can contain errors. Thereby we build error upon error, risking increasingly further deviations from the truth.

# Chapter 4

## Experimental Setup & Results

In order to be able to evaluate the different preprocessing methods and machine learning algorithms, and to determine if they are capable of solving our problem, various experiments will be performed. These experiments will be explained in Chapter 4.1, followed by the measures that are used to evaluate them in Chapter 4.1.2. The results of our experiments will then be presented in Chapter 4.2 and discussed in Chapter 4.3.

### 4.1 Experimental Setup

In Chapter 3 we discussed various preprocessing methods and two different machine learning algorithms. Some preprocessing steps were optional and most allowed us to use different parameters. The latter was also possible for the machine learning algorithms. These different options allow us to try various setups and investigate what results in the best-performing models. These setups will be discussed here

#### 4.1.1 Experimental parameters

The most prominent options available to us are the two machine learning algorithms and the prediction paradigms. These options form the basis of our experiments and, because we consider these the most important, each experiment will be performed four times; one time for each combination of machine learning algorithm and prediction paradigm. This increases the amount of experiments that need to be performed by a factor of 4 and thereby the required time to perform all experiments increases quickly for every extra option we want to consider. For this reason we decided not to investigate the performance of all different parameter combinations, but rather determine the effect that parameters have individually. We achieve this by choosing a baseline experiment and varying the value of one parameter at a time, allowing us to measure the effect of that parameter.

This baseline consists of a predetermined set of parameter values, which each was chosen by hand to try and create a ‘neutral’ experimental setup by selecting the most regular setting for each parameter. Subsequent experiments all have one changed parameter compared to the baseline to allow for a fair comparison. All tested parameter values were chosen to test a broad, though sensible, range of values for each parameter, based on empirical tests and commonly used settings, in order to determine which values lead to the best performance.

Parameter	Baseline values	Other values
Number of historic time-steps	4	[2, 3, 4, 5, 6]
Number of predicted time-steps	2	[2, 3, 4]
PCA features	No PCA	[No PCA, 0.75, 0.95]
Cap z-scores	No cap	[No cap, 1.75, 2.0, 2.25, 2.5]
Learning rate	0.025	[0.01, 0.025, 0.05]
Number of epochs	1000	[500, 1000, 2500, 5000, 10000]
Number of hidden units	0.5	[0.33, 0.5, 0.66]

**Table 4.1:** This table shows the different parameter values that are used for our experiments. The values for the baseline experiment are displayed separately. The top four parameters are preprocessing parameters while the bottom three are algorithmic parameters. Because both the MLP and CRBM algorithms are types of neural networks, they have common parameters that can be varied, although they will most likely have a different effect for each algorithm.

An overview of all experimental parameter values is given in Table 4.1. and an explanation of the different parameters and their possible values are given below:

- **Number of historic time-steps:** The number of historic time-steps that are used to make predictions.
- **Number of predicted time-steps:** Number of time-steps that will be predicted.
- **PCA features:** Indicates whether or not PCA features are added. A number denotes the minimum percentage of variance of the original data that the PCA features should account for.
- **Cap z-scores:** At what value the z-score will be capped to treat outliers.
- **Learning rate:** The learning rate that an algorithms utilises.
- **Number of epochs:** The amount of iterations a model will be trained for.
- **Number of hidden units:** The number of hidden units as factor of the number of input and output units.  $N \text{ hidden units} = (N \text{ input} + N \text{ output}) * \text{parameter value}$

As mentioned, all experiments, including the baseline experiment, will be performed four times, one time for each combination of algorithm and prediction paradigm. With a total of 20 parameter values that differ from the baseline values, this results in a total of  $20 * 4$ , plus 4 baseline experiments, minus 2 (see the second remark at the end of this chapter), for a total of 82 experiments that were performed.

From these experiments, the values of the parameters that individually performed best were selected and used for an experiment which combines them. While this does not substitute an exhaustive search of each combination of parameters, it does allow us to try a set of parameters that is not limited to one change with respect to the baseline. These parameters and the results will be presented in Chapter 4.2.

Finally, two remarks regarding the experiments have to be made. First, a large number of location and their respective devices were available to us: over 250 locations. If we were to use the data of all locations, some experiments would have taken days to complete. As a result we used a subset of the data of 10 locations per brand for a total of 30 locations. Second, the experiments that investigate the influence of additional PCA features were only performed in combination with the Single Step prediction paradigm. This is due to the fact that the Recursive paradigm uses data made in previous predictions. This means we either have to predict the regular features and the PCA values as well, or calculate the PCA values after each prediction. Predicting them propagates unnecessary prediction errors to future predictions and pollutes our evaluation measures. This option was therefore rejected. However, the required infrastructure to calculate the PCA features per prediction, such as accessing the required parameters and a mechanism to detect or indicate the presence of PCA features in the data, were unavailable. Despite this fact, we will still be able to investigate the effectiveness of additional PCA features from the experiments that utilise the Single Step paradigm.

### 4.1.2 Evaluation measures

This chapter deals with the different evaluation measures that are used to assess each experiment. However, before we discuss these measures in detail it, it may be helpful to restate the type of data that we are dealing with. The data we try to predict are counters, which are natural numbers<sup>1</sup>. The numbers that we actually predict are continuous, which can be rounded to a natural number. With this type of data there are various measures that we can calculate and use.

First of all, we can calculate a *confusion matrix*. A confusion matrix provides an overview of a model's performance, by displaying how many times each of the classes was classified right or wrong. In the latter case, it also displays how often it confused the target class with every other class. In our case we have two classes: the class *'event'*, which encodes if an event did occur at least once, and the class *'no event'*, which in turn encodes whether an event did not occur. In confusion matrix terminology, the *'event'* class is considered *'positive'* and the *'no event'* class is considered negative. This results in four fields within a confusion matrix:

- **True positives (TP)**: Number of times the model correctly classifies a positive instances as positive.
- **False positives (FP)**: Number of times the model wrongly classifies a negative instances as positive.
- **True negatives (TN)**: Number of times the model correctly classifies a negative instances as negative.
- **False negatives (FN)**: Number of times the model wrongly classifies a positive instances as negative.

---

<sup>1</sup>Multiple definitions of natural numbers are in use with regards to whether or not 0 is included. In this case we count 0 among the natural numbers.

Once we fill a confusion matrix by counting the right and wrong predictions of a model, we can calculate various measures, such as the *true positive rate* and *true negative rate*, which provide a measure of how many times both classes were correctly predicted. A common practice is to calculate the true positive and *false positive* rates to plot a *Receiver Operating Characteristic* (ROC) curve. Such a curve provides an indication of how reliable a model is when it classifies an instance as being positive, the True positives (TP). However, for this project we are primarily concerned with maximising the amount of times a model is correct when an instance is classified as ‘positive’. We therefore use the following measures for the assessment of our models’ performance:

- **True Positive rate** (TPR): The percentage of all positive instances that are classified as being a positive instance. Calculated using the following formula:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **Precision:** The percentage of the positively classified instances that are truly positive. This can be calculated using the equation below:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **True Negative rate** (TNR): The percentage of all negative instances that are classified as being a negative instance. Calculated using the following formula:

$$\text{TNR} = \frac{\text{TN}}{\text{TN} + \text{FP}}$$

- **Accuracy:** The percentage of correct predictions from all predictions together. This percentage can be calculated with the following equation:

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

Together, the True Positive rate and Precision give an indication of the ‘quality’ of a model’s prediction when it predicts that an instance is positive. Along with the True Negative Rate and the Accuracy, these four measures provide an overview of the model’s performance with an emphasis on positive instances.

It was briefly mentioned that the predicted continuous numbers, in combination with a threshold, are used to determine when it is predicted that an event will occur. A predicted value below the threshold will be interpreted as an event not occurring, i.e. a negative instance, while a value above it will be interpreted as an event that does occur, i.e. a positive instance. Different values of this threshold can be used to make a model more liberal or strict when it judges a prediction. A lower threshold will cause it to sooner judge an instance as being positive, while a model requires a more convincing prediction when a higher threshold is selected. Varying this threshold therefore has direct consequences for each confusion matrix and the measures that we selected, as they are based on these matrices. Doing this allows us to systematically vary the threshold and plot the resulting values of each measure, which gives a great overview of the performance of each model and the trade-off of lowering or increasing the threshold. The results presented in 4.2 will contain such plots.



Finally, the format of the real and predicted data allows for the calculation of the well known Mean Squared Error (MSE) measure for the trained models. This measure is an error measure that allows for comparison between different models and for which lower values are better. The MSE can be calculated using the following equation:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (X_i - X'_i)^2$$

Where  $X$  is the set of all vectors,  $X_i$  and  $X'_i$  denote the true and predicted vectors respectively and  $n$  is the total number of vectors. This measure is used to compare models that are created with different sets of parameter values.

## 4.2 Results

This chapter will present the results of the performed experiments, which are obtained using 5-fold cross-validation. Due to the large number of experiments, only the results of the baseline and optimised experiments will be presented. The remainder of the results can be found in Appendix B and C instead, which display the obtained MSE scores and the plotted evaluation measures respectively. All reported MSE values are averaged over the results of the individual models that were generated for a particular experiment. Their respective standard deviations are included between parentheses within the tables.

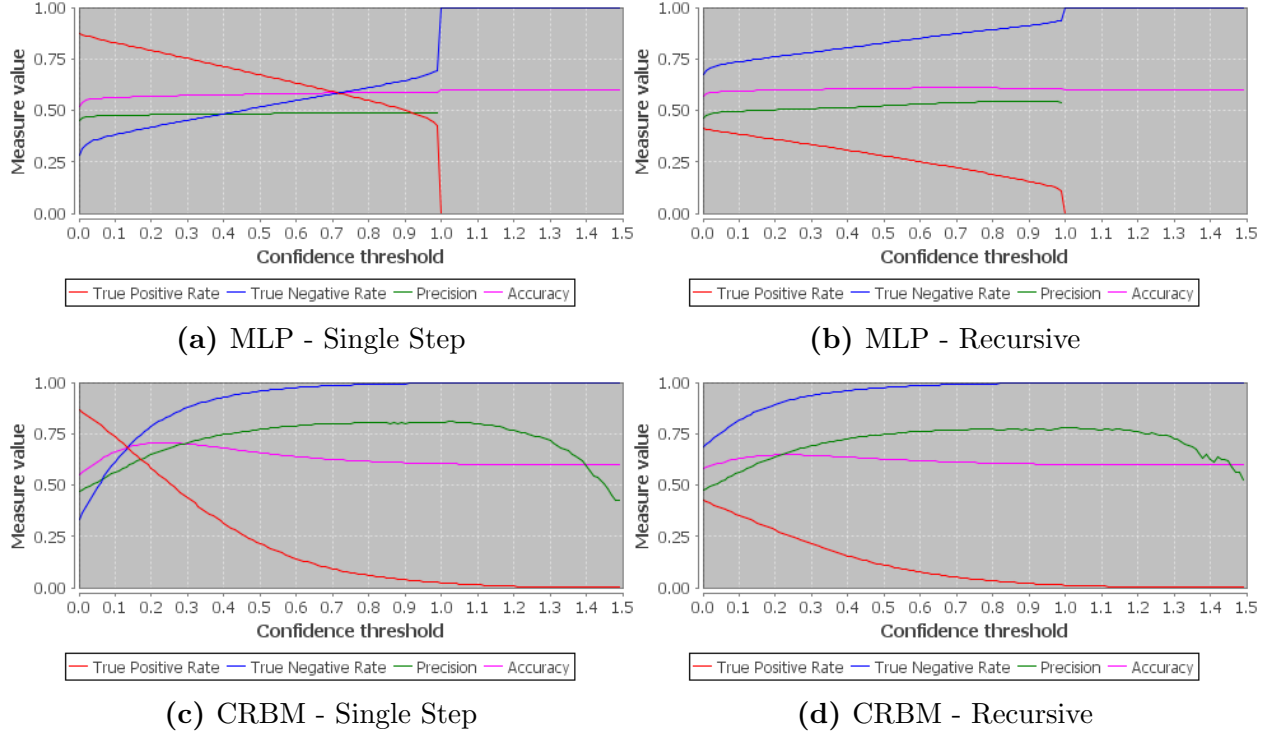
For the baseline experiments, which created models using the parameter values shown in Table 4.1, the MSE scores can be found in Table 4.2 and the corresponding evaluation measures can be found in Figure 4.1.

	MLP Single Step	MLP Recursive	C-RBM Single Step	C-RBM Recursive
Baseline	0.248 (0.252)	0.158 (0.181)	0.187 (1.458)	0.112 (0.055)

**Table 4.2:** The resulting MSE values (with corresponding Standard Deviations between parentheses) for the baseline experiments for each combination of algorithm and prediction paradigm.

The columns of the MSE-score table each represent an algorithm and prediction paradigm combination. From this table we can then determine that for both algorithms the Recursive paradigm, in comparison to the Single Step paradigm, achieves the lowest MSE scores, along with a smaller standard deviation. The best score overall is achieved by the CRBM - Recursive combination and the worst score is achieved by the MLP - Single Step combination.

However, when we consider Figure 4.1, which displays the evaluation measures for the baseline experiments, we can observe a few interesting differences. While the CRBM - Recursive combination achieved the best MSE scores, it is the CRBM - Single Step combination that is able to achieve the highest Accuracy of about 0.70 (at a confidence threshold of about 0.2) versus a rough 0.65 for the CRBM - Recursive combination. When we consider the figure from a broader perspective, we can observe that the Precision and Accuracy differ very little between the two paradigms, but that True positive rate and True negative rate almost seem to switch their starting points within each graph. This indicates that the Single Step paradigm seems to favour making positive predictions (by predicting higher values for counters) and that the Recursive paradigm does the opposite by predicting lower values, thereby achieving a higher True negative rate.



**Figure 4.1:** Results of the baseline experiments for each combination of algorithm and prediction paradigm.

Furthermore, what Figure 4.1 also displays prominently, is a difference between the two algorithms. For the CRBM algorithm, each measure displays a relatively smooth curve all the way to a threshold of 1.5, while the situation is a bit different for the MLP algorithm, which shows an interruption of the trend at 1.0. This is due to the Sigmoid activation function which the MLP algorithm utilises (see Chapter 3.2.1), which has an output limit of 1.0. This property thereby imposes a natural maximum threshold, which is reflected in the plots of the measures.

An inspection of the results of the other experiments, which are displayed in Appendices B and C, show similar results, but a few anomalies can also be observed.

First, Table B.3, which contains the results of the experiments investigating the effect of the number of predicted time-steps, displays high MSE-values in the ‘CRBM Single Step’ column for parameter values deviating from the baseline. These abnormal values are not the result of miscalculations or erroneous models resulting from external problems that occurred during training, but rather of predictions that merely deviate a lot from their true values. This was concluded when the problem persisted after retraining the models and is further supported by the fact that this problem only occurs for the CRBM - Single Step combination. The latter reason rules out any problems with the data, as the same data was used for training the models of the remaining algorithm and paradigm combinations. A closer inspection of the predicted data revealed that the problem is tied to a small subset of all devices for which the predictions deviated a lot from the real data, while the predictions for the majority of devices were according to expectations. Unfortunately, we can offer no satisfactory explanation for this problem and therefore these results are not considered in the process of optimising parameters for the CRBM - Single Step combination.

	MLP Single	MLP Recursive	C-RBM Single	C-RBM Recursive
Best achieved MSE	0.157 (0.151)	0.088 (0.086)	0.093 (0.248)	0.090 (0.051)

**Table 4.3:** The best achieved MSE values for the unoptimised experiments.

Parameter	MLP Single Step	MLP Recursive	C-RBM Single Step	C-RBM Recursive
Number of historic time-steps	2	2	3	4
Number of predicted time-steps	2	2	2	2
PCA features	No PCA	No PCA	No PCA	No PCA
Cap z-scores	No cap	No cap	No cap	No cap
Learning rate	0.025	0.01	0.01	0.01
Number of epochs	500	500	10000	10000
Number of hidden units	0.33	0.33	0.33	0.33

**Table 4.4:** Parameter values that resulted in the best performance for previous experiments.

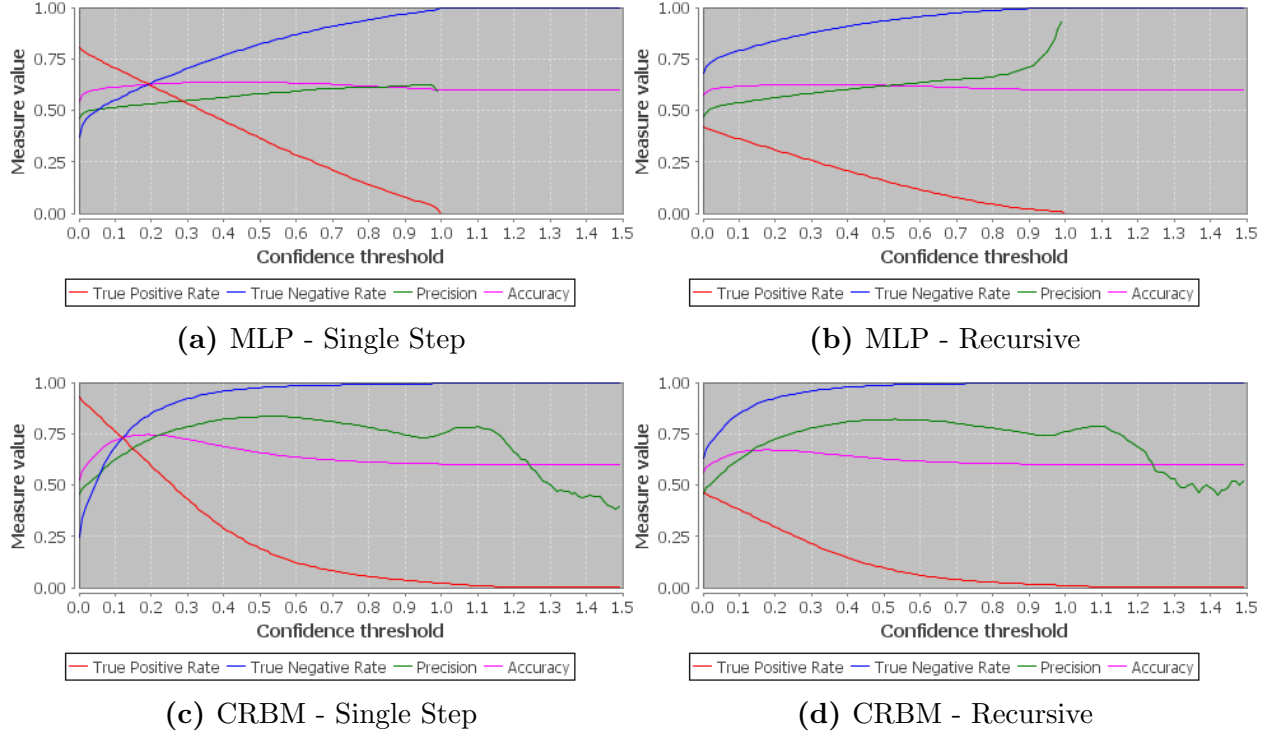
Second, the PCA results for the ‘CRBM Single Step’ combination, as displayed in Table B.4, also suffer from the above-mentioned problem. However, the MLP variant does not, which still allows us to draw conclusions with regards to the addition of PCA parameters. Furthermore, the accuracy of the CRBM measures displayed in Figures C.8 and C.9, differ from the baseline CRBM measures to such an extent, that it allows us to conclude that the addition of PCA features is not helpful.

Now that all results are available to us and we disregarded problematic results, we can determine the best achieved MSE scores for each combination of algorithm and prediction paradigm. These scores are displayed in Table 4.3 and were obtained using a hidden unit ratio of 0.33 for the MLP algorithm (for both prediction paradigms) and 10,000 epochs for the C-RBM algorithm (again, for both paradigms).

Furthermore, we can use all results to determine the best parameter values and combine these to create a new experiment. This experiment allows us to investigate whether the combination of these values leads to a better performance than what we already achieved. The parameter values which achieved the best performances are displayed in Table 4.4 and the resulting MSE scores and performance measures of this experiment are displayed in Table 4.5 and Figure 4.2 respectively.

	MLP Single Step	MLP Recursive	C-RBM Single Step	C-RBM Recursive
Optimised settings	0.155 (0.200)	0.131 (0.115)	0.095 (0.102)	0.099 (0.070)

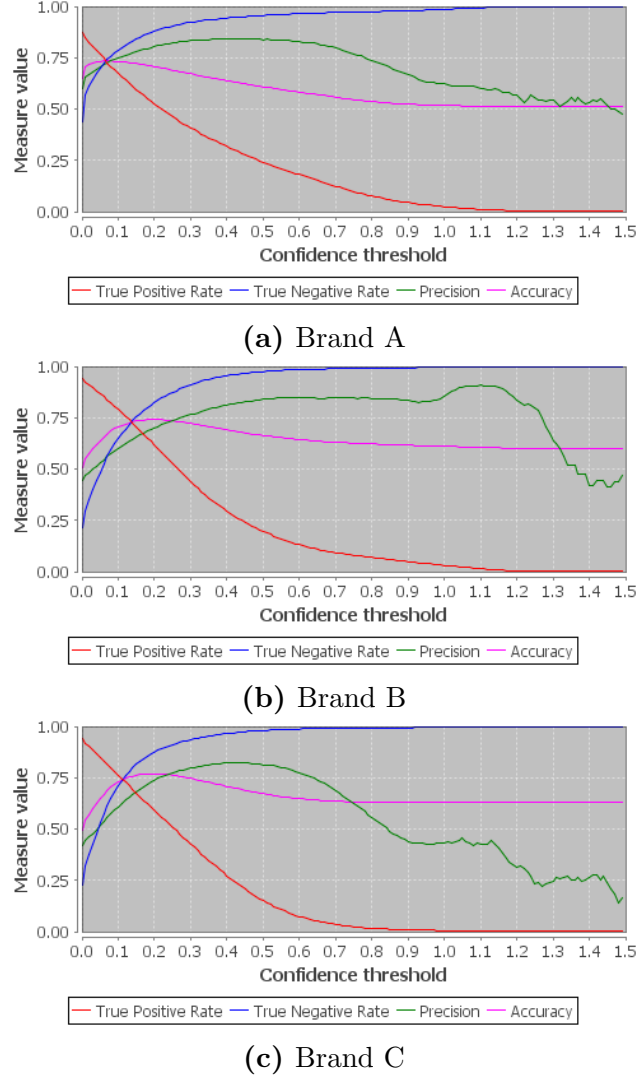
**Table 4.5:** The MSE values for the optimised experiments.



**Figure 4.2:** Results of the experiments with the optimised parameter values for each algorithm - prediction paradigm combination.

When comparing the new MSE scores in Table 4.5 to the previous best result in Table 4.3, it can be observed that none of the new MSE scores is actually better, although the differences are small for all but the MLP - Recursive combination. Standard deviations also increased, except for the CRBM - Single Step combination, which shows a standard deviation that is less than half the standard deviation of the previous best MSE score. However, the performance measures do seem to have improved, featuring a higher precision and accuracy in all cases, if only slightly. Overall the CRBM - Single Step combination achieved the best results, both in terms of MSE scores and evaluation measures.

These optimised results can now be used to investigate our model's capacity for generalisation, by extracting the results for the three different brands and comparing them. The evaluation measures for each brand are displayed in Figure 4.3. Apart from the rather apparent differences for the Precision measure between each brand, the other plotted measures are very similar. Furthermore, an accuracy of about 0.75 is achieved for all brand. The threshold values at which these accuracy scores are obtained, also yield precision scores of 0.70 to 0.75. The respective MSE scores of these results are displayed in Table 4.6, which show us that the models' performance of Brand A & B is very similar, but a bit worse for Brand C. This is actually rather interesting, as the Accuracy evaluation measure has the highest peak for Brand C, while it intuitively makes more sense if a model with a low MSE value also has a higher Accuracy score.



**Figure 4.3:** The evaluation measures for each brand from the CRBM - Single Step experiment with optimal parameter values.

	Brand A	Brand B	Brand C
CRBM - Single Step optimal parameter values	0.087 (0.040)	0.081 (0.044)	0.118 (0.072)

**Table 4.6:** MSE scores for each brand from the CRBM - Single Step experiment with optimal parameter values.

## 4.3 Discussion of results

The results allow for a number of interesting observations with regards to the machine learning algorithms, prediction paradigms, the different experimental parameters and the three brands.

**Evaluating the machine learning algorithms** With identical parameter values, the MLP algorithm obtains worse MSE scores and is not able to achieve the same levels of the Accuracy and Precision evaluation measures as the CRBM algorithm. However, in combination with the Recursive prediction paradigm, the MLP algorithm is able to achieve the lowest MSE-score overall in the '0.33 hidden unit ratio' experiment (0.088), but this does not lead to impressing evaluation measures (see Figure C.20). Instead, the CRBM algorithm is able to achieve a similar MSE score of 0.093 in the '10,000 epochs' experiment and the best evaluation measures in the optimised setting in combination with the Single Step prediction paradigm. Therefore it can be concluded that the CRBM algorithm is better suitable for this application.

**Evaluating the prediction paradigms** Overall the Recursive prediction paradigm achieved the lowest MSE scores, along with lower standard deviations than the Single Step paradigm. This also holds true for the best achieved average MSE scores. Where evaluation measures are concerned, similar values are obtained for both prediction paradigms for the Accuracy and Precision measures, although, in combination with the CRBM algorithm, the Single Step produces substantially higher peaks for the Accuracy measure. This is actually rather interesting, since one would not expect such a difference for models obtaining similar MSE scores. Furthermore, there is a clear difference for the True positive rate and True negative rate measures between the two paradigms. The Single Step paradigm favours the True positive rate and the Recursive paradigm favours the True Negative rate. Considering we are more interested in when an event is predicted (so that proactive measures can be taken should the event warrant such actions), rather than predicting that nothing is going to happen, combined with the higher Accuracy values, appoints the Single Step paradigm as the preferred method for making predictions.

### **Evaluation of experimental parameters for the CRBM - Single Step combination**

Our experiments allowed us to investigate the effect of the different experimental parameters and we can draw the following conclusions from the MSE values and various measures as displayed in Appendices B and C:

- Varying the number of historic and predicted time-steps does not seem to influence performance much. Lowering the number of historic time-steps to 3, which results in a historic:predicted time-step ratio of 3:2 achieves the best results.
- Enforcing maximum allowed z-scores has a negative influence on the MSE scores and can increase their respective standard deviations, while having mixed effects on evaluation measures. Enforcing no maximum leads to the lowest MSE scores and standard deviation, rendering it the most reliable choice.
- It is in no way beneficial to add additional PCA features for this application, as doing so leads to worse scores for the Accuracy, True positive rate and True negative rate measures.

- A low learning rate of 0.01 leads to the most favourable results, for both the MSE scores and evaluation measures, making a slow approach to learning the best approach.
- The longer the training phase lasts, the better the CRBM algorithm seems to learn to predict the data. Training for 10,000 epochs on our relatively small dataset produced the best results with no apparent signs of overfitting.
- Creating less complex models that use a number of hidden units ratio of 0.33 as a factor to the number of input and output units together, lead to marginally better results than ratios of 0.5 or 0.66.

**Evaluating the performance for brands** A similar performance was achieved across the three different brands. The lowest MSE values were achieved by brands A & B, while brand C achieved the highest Accuracy score. Despite these differences, the results for the three brands, as achieved by the CRBM - Single Step combination using the optimised parameters values, were sufficiently similar to be able to conclude that the current setup is able to generalise over different brands very well.

### **A general remark with respect to the statistical significance of obtained results**

Within this chapter various conclusions have been drawn with regards to which methods were judged to perform better, or worse, with respect to each other. Such statements were made without reporting any statistical results that indicate whether the differences between compared methods are significant or not. An example for which the results of a statistical test could be interesting, are the results obtained by the ‘MLP - Recursive’ combination in the ‘0.33 hidden unit’ experiment, which obtained an average MSE score of 0.088, versus the ‘CRBM - Single Step’ combination in the ‘10,000 epochs’ experiment, which achieved an average MSE score of 0.093. These results are similar to the extent that tests which determine whether the difference is significant are warranted, as it could be concluded that the performance of both combinations, with their respective parameter values, are similar and that one does not differ that much from the other. However, such a conclusion does not consider the other evaluation measures (the True positive / negative rates and Precision and Accuracy scores), which show values that are far less similar than the MSE scores. Between these two experiments, the remaining four evaluation measures differ to such an extent that the MSE scores are not of importance, let alone whether the difference is statistically significant.

For every relevant comparison of experiments which display similar evaluation measures, there are other evaluation measures which exhibit differences so convincing, that we are justified to draw a conclusion with regards to which setup is preferred, without performing any statistical tests. Relevant comparisons are those which determine the differences between algorithms, prediction paradigms or parameter values in contrast to the baseline.





# Chapter 5

## Discussion

The aim of this project was to create a generic system, capable of predicting the events that are generated by systems present in parking garages, regardless of the brand or exact type of such systems. In this chapter we will review the complete project with respect to our goals, as defined in Chapter 1.4, and assess to what extent they were accomplished. We will furthermore consider the subject future research could focus on and, finally, provide an overall conclusion of the project.

### 5.1 Assessing generic event prediction capabilities

Two main questions guided this work, which were defined in Chapter 1.4. The first question asked us whether existing methods and algorithms from related fields, such as event predicting and time-series forecasting, could be applied in our context; the predicting of events generated by the systems present in parking garages. The second question asked whether it would be possible to create a generic event-prediction model, capable of making predictions for different types of systems from varying manufacturers.

Such a generic model, based either on existing or novel methods, had to be capable of making accurate predictions in each possible situation. Our results show that a setup using a Conditional Restricted Boltzmann Machine, in combination with the Single Step prediction paradigm, which predicts multiple future time periods within a single prediction iteration, is able to achieve this goal. With this setup we were able to build models that predict the occurrences of events within future time-steps with an average accuracy of around 75%, regardless of machine manufacturer, as demonstrated in Figures 4.2c and Figure 4.3 in Chapter 4.2. At these accuracy values, precision scores of 70% to 75% were also obtained. These results demonstrated the model's capabilities to generalise over different types of machines from multiple manufacturers, as well as the possibility to apply the existing methods and algorithms, and thereby answer our main questions. The results also imply that this system could be extended to allow for devices from other manufacturers, provided that the information reported by their devices can be transformed to the generic format as described in Chapter 3.1.

While the setup's demonstrated generalisation capabilities were one of the main goals of this work, the consequences of the obtained accuracy and precision scores, and the implied value for aiding the process of planning proactive maintenance, have to be taken into consideration. These scores imply that 25% to 30% of all predicted events actually will not occur and that any taken actions, based on faulty predictions, appear to be unnecessary.

However, while such proactive actions may not prevent immediate failures, performing maintenance on machines still increases their longevity and might postpone the moment at which the failures actually would occur, whether the events on which such decision are based were correctly predicted or not. Next to that, the act of performing premature maintenance, based on a wrong prediction, is not that much different from the original situation in which regular maintenance might be performed too often.

Furthermore, the results were obtained with parametric values that were only optimised to a certain length, due to computational limits, and better performance might yet be achieved if they are optimised further. Next to that, we did not filter for events that are either very rare, and therefore hard to predict within the used time-frame of four weeks, or events that have no long-term precursors, such as immediate externally induced events, e.g. a fire or vandalism. These events can be considered to be outside the scope of predictable events and filtering them out could reveal higher scores for the remaining events. However, due to a lack of event descriptions we were unable to perform such filtering during this project.

## 5.2 Future research

While we were able to achieve our goal of creating a generic setup, capable of generation models for the prediction of event time-series, numerous options are available which might improve the predictive performance of the models further or the setup's practical applicability.

The setup which was created for this work, generates a model for every individual device. Although it was shown to perform its task successfully, such models do not take their environment into account in the sense that no information is shared between the models of all devices located within the same parking garage. A model that monitors and predicts events for all devices together, might be able to determine and exploit any interaction between the behaviour of devices and use these relations to make more reliable predictions.

Secondly, the current set of parameter values for each experiment were hand-picked, but, even though they were picked with careful consideration, there is no guarantee that the optimal combination of values was selected or that the optimal parameter values themselves were even an option. Finding the best parameter values is a challenge in itself, which might be solved through the use of a genetic algorithm.

Third, an effort can be made to determine whether the current scores can be improved, or if the best possible scores already have been obtained. This could be achieved by performing a number of experiments using random parameter values. The required number of experiments would have to be statistically determined. If none of these experiments achieve a better performance than what is achieved by the current best set of parameter values, it can safely be assumed that a (near) optimum score indeed is achieved.

Finally, another interesting line of research could look into a different type of generalisation. The current project's aim was to create a setup that was able to build reliable models for individual devices, regardless of a device's brand or system type, based on data of that specific device gathered over a period of time. However, it would also be very valuable if a setup could be created that builds generalised models for each type of device, based on the existing data of all devices of that specific type together.

Such a setup would allow for the immediate deployment of basic predictive models to newly installed machines, without the need of having to gather usage and behavioural data of that specific device first. Once such data would become available, such basic models could be refined. However, this would require an accurate list of the different device types and to what type each individual device belongs to, but such a list is currently unavailable. Even so, such a list might be compiled through clustering, based on which events occur for each device.

## 5.3 Conclusion

We set out to build a generic setup that should be capable of producing reliable predictive models for the prediction of events generated by devices in parking garages, regardless of their type or manufacturer. With a setup using the Single Step prediction paradigm and the Conditional Restricted Boltzmann Machine algorithm, we achieved positive results with an average maximum accuracy of 75%. Furthermore, results showed a similar performance for each of the three evaluated brands.

While an accuracy of 75% is not a performance that is generally deemed satisfactory, this setup is judged to perform sufficiently well to the extent that it will be applied in a real world setting where it will aid the process of deciding when proactive-oriented maintenance should be applied. Combined with this knowledge, the obtained results therefore demonstrate the success of our setup and the achievement of our goal to create a generic, predictive model.



# Bibliography

- [1] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.
- [2] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines\*. *Cognitive science*, 9(1):147–169, 1985.
- [3] Mark O Afolabi and Olatoyosi Olude. Predicting stock prices using a hybrid kohonen self organizing map (som). In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 48–48. IEEE, 2007.
- [4] Nesreen K Ahmed, Amir F Atiya, Neamat El Gayar, and Hisham El-Shishiny. An empirical comparison of machine learning models for time series forecasting. *Econometric Reviews*, 29(5-6):594–621, 2010.
- [5] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. Machine learning strategies for time series forecasting. In *Business Intelligence*, pages 62–77. Springer, 2013.
- [6] George EP Box, Gwilym M Jenkins, and Gregory C Reinsel. *Time series analysis: forecasting and control*. John Wiley & Sons, 2013.
- [7] Arthur E Byson Jr and Yu-Chi Ho. Applied optimal control. *Ginn & Co., Waltham, Toronto, London*, 1969.
- [8] Sean D Campbell and Francis X Diebold. Weather forecasting for weather derivatives. *Journal of the American Statistical Association*, 100(469), 2005.
- [9] Hsin Chen and Alan F Murray. Continuous restricted boltzmann machine with an implementable training algorithm. In *Vision, Image and Signal Processing, IEE Proceedings-*, volume 150, pages 153–158. IET, 2003.
- [10] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.
- [11] David Harris, W Kuhrman, Lisa Schmit, and Jerry Sychra. Method for predicting machine or process faults and automated system for implementing same, 2001.
- [12] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

- [13] CG Kilsby, PSP Cowpertwait, PE O’connell, and PD Jones. Predicting rainfall statistics in england and wales using atmospheric circulation variables. *International Journal of Climatology*, 18(5):523–539, 1998.
- [14] Kyoung-jae Kim. Financial time series forecasting using support vector machines. *Neurocomputing*, 55(1):307–319, 2003.
- [15] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.
- [16] Jay Lee, Jun Ni, Dragan Djurdjanovic, Hai Qiu, and Haitao Liao. Intelligent prognostics tools and e-maintenance. *Computers in industry*, 57(6):476–489, 2006.
- [17] KH Lee and GS Jo. Expert system for predicting stock market timing using a candlestick chart. *Expert Systems with Applications*, 16(4):357–364, 1999.
- [18] GI Marchuk. Numerical methods of weather forecasting. Technical report, DTIC Document, 1970.
- [19] James L McClelland, David E Rumelhart, PDP Research Group, et al. Parallel distributed processing. *Explorations in the microstructure of cognition*, Volumes 1, 2 & 3, 1986.
- [20] Cynthia Rudin, David Waltz, Roger N Anderson, Albert Boulanger, Ansaf Salieb-Aouissi, Maggie Chow, Haimonti Dutta, Philip N Gross, Bert Huang, Steve Ierome, et al. Machine learning for the new york city power grid. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(2):328–345, 2012.
- [21] Ramendra K Sahoo, Adam J Oliner, Irina Rish, Manish Gupta, José E Moreira, Sheng Ma, Ricardo Vilalta, and Anand Sivasubramaniam. Critical event prediction for proactive management in large-scale computer clusters. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 426–435. ACM, 2003.
- [22] Felix Salfner, Maren Lenk, and Mirosław Malek. A survey of online failure prediction methods. *ACM Computing Surveys (CSUR)*, 42(3):10, 2010.
- [23] W Sarle. Neural networks frequently asked questions. 1997.
- [24] Murali Shanker, Michael Y Hu, and Ming S Hung. Effect of data standardization on neural network training. *Omega*, 24(4):385–397, 1996.
- [25] Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. 1986.
- [26] Graham W Taylor, Geoffrey E Hinton, and Sam T Roweis. Modeling human motion using binary latent variables. In *Advances in neural information processing systems*, pages 1345–1352, 2006.
- [27] Graham W Taylor, Geoffrey E Hinton, and Sam T Roweis. Two distributed-state models for generating high-dimensional time series. *The Journal of Machine Learning Research*, 12: 1025–1068, 2011.

- [28] Y-F Wang. Predicting stock price using fuzzy grey prediction system. *Expert Systems with Applications*, 22(1):33–38, 2002.
- [29] Gary M Weiss and Haym Hirsh. Learning to predict rare events in event sequences. In *KDD*, pages 359–363, 1998.
- [30] PJ Werbos. Beyond regression: New tools for predictions and analysis in the behavioral science. cambridge, ma, itd, 1974.
- [31] Leon Li Wu, Gail E Kaiser, Cynthia Rudin, David L Waltz, Roger N Anderson, Albert G Boulanger, Ansaf Salieb-Aouissi, Haimonti Dutta, and Manoj Pooleery. Evaluating machine learning for improving power grid reliability. 2011.
- [32] Zimin Max Yang, Dragan Djurdjanovic, and Jun Ni. Maintenance scheduling in manufacturing systems based on predicted machine degradation. *Journal of Intelligent Manufacturing*, 19(1): 87–98, 2008.
- [33] Matthew D Zeiler, Graham W Taylor, Nikolaus F Troje, and Geoffrey E Hinton. Modeling pigeon behavior using a conditional restricted boltzmann machine. In *ESANN*, 2009.





# Appendix A

## Test environment interface images

Experiment parameters

Cross-validation parameters

☒ Use cross-validation    N folds: 5

☐ Custom fold size (%)    20

Forecasting algorithm parameters

Classifier: Multilayer Perceptron

Stopping criterion: 500.0    Epochs

N hidden units: auto

Learning rate: 0.05

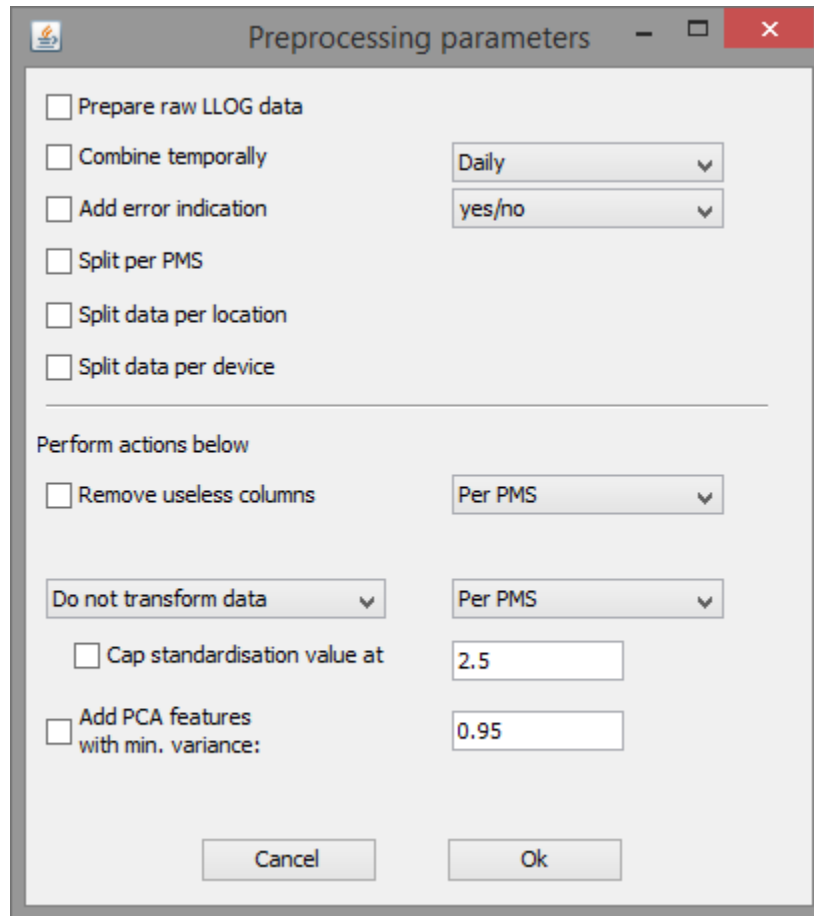
Momentum: 0.1

Prediction options

Prediction paradigm: Single-step

Cancel    Ok

**Figure A.1:** The ‘Experiment parameter’ window showing possible settings with regards to cross-validation, the selection of a machine learning algorithm and its parameters and the prediction paradigm that should be used when making predictions.



The 'Preprocessing parameters' window contains the following settings:

- ☐ Prepare raw LLOG data
- ☐ Combine temporally: Daily
- ☐ Add error indication: yes/no
- ☐ Split per PMS
- ☐ Split data per location
- ☐ Split data per device

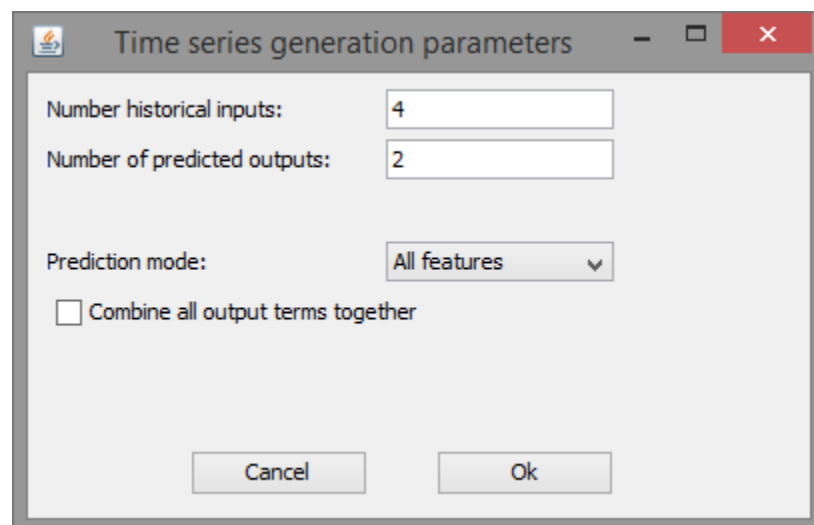
---

Perform actions below

- ☐ Remove useless columns: Per PMS
- Do not transform data: Per PMS
- ☐ Cap standardisation value at: 2.5
- ☐ Add PCA features with min. variance: 0.95

Buttons: Cancel, Ok

**Figure A.2:** The ‘preprocessing parameter window’ showing the various possible preprocessing methods and, where applicable, their parameters.

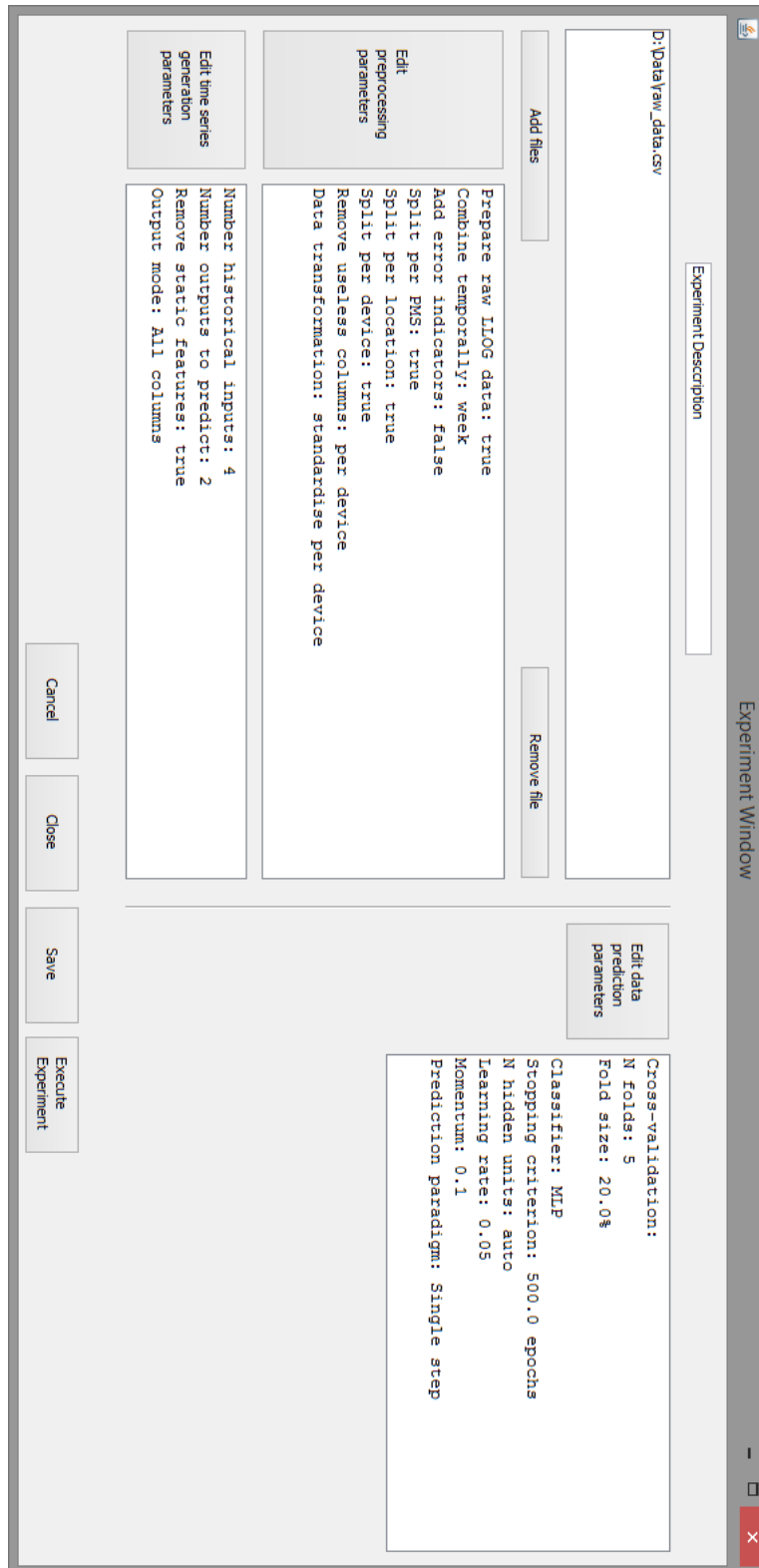


The 'Time series generation parameters' window contains the following settings:

- Number historical inputs: 4
- Number of predicted outputs: 2
- Prediction mode: All features
- ☐ Combine all output terms together

Buttons: Cancel, Ok

**Figure A.3:** The ‘time series generation parameter’ window showing the possible options for generating time series.



**Figure A.4:** The experiment window from which the interfaces for setting the different parameters, as shown in the previous images, are accessible. It also shows a textual representation of the selected parameters. At the top left it is possible to select the data that should be used when the pipeline starts.

# Appendix B

## Experiment results: tables reporting MSE values

The tables shown in this appendix display the MSE values, along with their respective standard deviations in parentheses, that were obtained for each experiment. Within these tables, the baseline parameters are marked with an asterisk (\*) and the best average MSE values of each combination of algorithm and prediction paradigm are marked in bold for convenience. Each row in a table represents an experimental condition while each column represents an algorithm and prediction paradigm combination. The appendix itself is ordered by experimental parameter.

### B.1 Baseline experiment

	MLP	MLP	C-RBM	C-RBM
	Single Step	Recursive	Single Step	Recursive
Baseline	0.248 (0.252)	0.158 (0.181)	0.187 (1.458)	0.112 (0.055)

**Table B.1:** The resulting MSE values (with corresponding Standard Deviations in parentheses) for the baseline experiments for each combination of algorithm and prediction paradigm. Re-included for convenience.

## B.2 Number of historic time-steps

Number historic time-steps	MLP Single Step	MLP Recursive	C-RBM Single Step	C-RBM Recursive
2	<b>0.163</b> (0.144)	<b>0.109</b> (0.094)	0.179 (1.110)	0.122 (0.071)
3	0.202 (0.204)	0.117 (0.118)	<b>0.160</b> (0.916)	0.121 (0.074)
4*	0.248 (0.252)	0.157 (0.181)	0.187 (1.458)	<b>0.112</b> (0.055)
5	0.283 (0.290)	0.199 (0.241)	0.314 (3.901)	0.120 (0.259)
6	0.317 (0.328)	0.239 (0.282)	0.277 (2.667)	0.128 (0.605)

**Table B.2:** Resulting MSE values for the experiments that vary the number of historic time-steps.

## B.3 Number of predicted time-steps

Number predicted time-steps	MLP Single Step	MLP Recursive	C-RBM Single Step	C-RBM Recursive
2*	<b>0.248</b> (0.252)	0.157 (0.181)	0.187 (1.458)	0.112 (0.055)
3	0.294 (0.282)	0.154 (0.180)	1.987 (10.699)	0.111 (0.056)
4	0.325 (0.301)	<b>0.152</b> (0.178)	7.469 (23.349)	<b>0.109</b> (0.056)

**Table B.3:** Resulting MSE values for the experiments that vary the number of predicted time-steps.

## B.4 PCA features

PCA features	MLP Single Step	MLP Recursive	C-RBM Single Step	C-RBM Recursive
<i>No PCA</i> *	<b>0.248</b> (0.252)	0.157 (0.181)	<b>0.187</b> (1.458)	0.112 (0.055)
0.75	2.007 (10.437)	N.A.	9.880 (52.554)	N.A.
0.95	2.440 (11.974)	N.A.	10.373 (54.542)	N.A.

**Table B.4:** Resulting MSE values for the experiments that test the influence of additional PCA features. Not applicable for the Recursive paradigm.

## B.5 Cap z-scores

Cap z-scores	MLP Single Step	MLP Recursive	C-RBM Single Step	C-RBM Recursive
1.75	0.269 (0.245)	0.174 (0.183)	0.309 (2.638)	0.128 (0.059)
2.0	0.263 (0.248)	0.167 (0.179)	0.296 (3.089)	0.122 (0.055)
2.25	0.258 (0.251)	0.162 (0.179)	0.246 (2.369)	0.120 (0.060)
2.5	0.254 (0.249)	0.161 (0.182)	<b>0.183</b> (1.541)	0.115 (0.055)
<i>No cap*</i>	<b>0.248</b> (0.252)	<b>0.157</b> (0.181)	0.187 (1.458)	<b>0.112</b> (0.055)

**Table B.5:** Resulting MSE values for the experiments that test the influence of capping the z-scores.

## B.6 Learning rate

Learning rate	MLP Single Step	MLP Recursive	C-RBM Single Step	C-RBM Recursive
0.01	0.250 (0.254)	<b>0.155</b> (0.181)	<b>0.178</b> (1.311)	<b>0.111</b> (0.054)
<i>0.025*</i>	<b>0.248</b> (0.252)	0.157 (0.181)	0.187 (1.458)	0.112 (0.055)
0.05	0.249 (0.255)	0.157 (0.184)	0.189 (1.906)	<b>0.111</b> (0.053)

**Table B.6:** Resulting MSE values for the experiments that test a model’s performance with varying learning rates.

## B.7 Epochs

Number of epochs	MLP Single Step	MLP Recursive	C-RBM Single Step	C-RBM Recursive
500	<b>0.242</b> (0.259)	<b>0.145</b> (0.182)	0.271 (2.149)	0.125 (0.056)
<i>1000*</i>	0.248 (0.252)	0.157 (0.181)	0.187 (1.458)	0.112 (0.055)
2500	0.261 (0.252)	0.168 (0.182)	0.173 (1.837)	0.098 (0.058)
5000	0.266 (0.249)	0.177 (0.182)	0.123 (1.221)	0.093 (0.053)
10,000	0.273 (0.246)	0.183 (0.183)	<b>0.093</b> (0.248)	<b>0.090</b> (0.051)

**Table B.7:** Resulting MSE values for the experiments that test a model’s performance with varying training durations.

## B.8 Number of hidden units

Number of hidden units	MLP Single Step	MLP Recursive	C-RBM Single Step	C-RBM Recursive
0.33	<b>0.157</b> (0.151)	<b>0.088</b> (0.086)	<b>0.101</b> (0.047)	<b>0.107</b> (0.055)
$0.5^*$	0.248 (0.252)	0.157 (0.181)	0.187 (1.458)	0.112 (0.055)
0.66	0.319 (0.314)	0.231 (0.260)	0.453 (3.879)	0.116 (0.056)

**Table B.8:** Resulting MSE values for the experiments that test a model’s performance with varying numbers of hidden units.



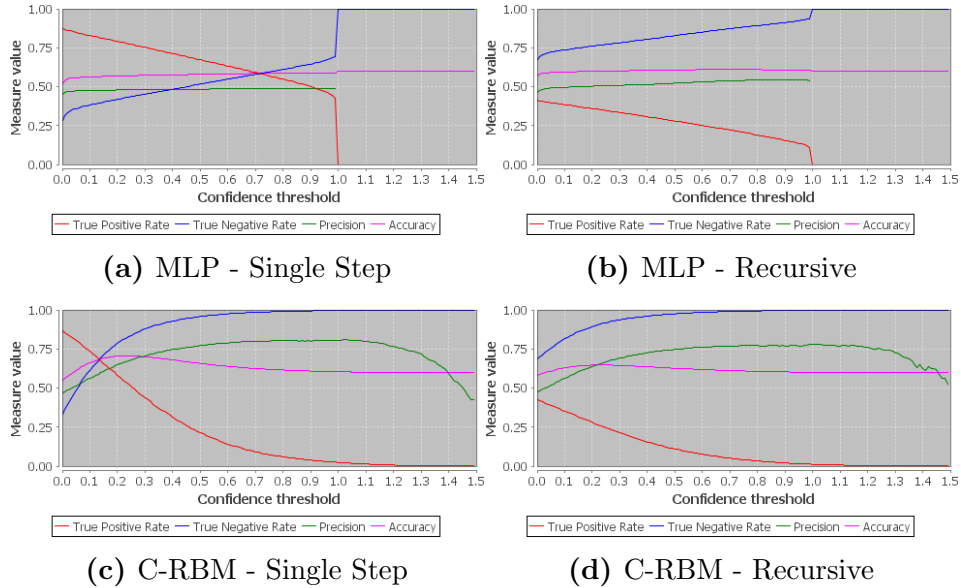
# Appendix C

## Experiment results: graphs of evaluation measures

This appendix shows the graphs displaying all evaluation measures (see Chapter 4.1.2 for an explanation) for each experiment that has been performed. The x-axis of the plots show the *confidence threshold*, which is used to round predicted numbers and determine whether an event has occurred or not, and is plotted against the measure value on the y-axis.

The appendix is ordered by experiment parameter and for each investigated parameter value four graphs are displayed; one for each algorithm and prediction paradigm combination.

### C.1 Baseline experiment



**Figure C.1:** Results of the baseline experiments for each combination of algorithm and prediction paradigm. Re-included for convenience.

## C.2 Number of historic time-steps

### C.2.1 2 historic time-steps

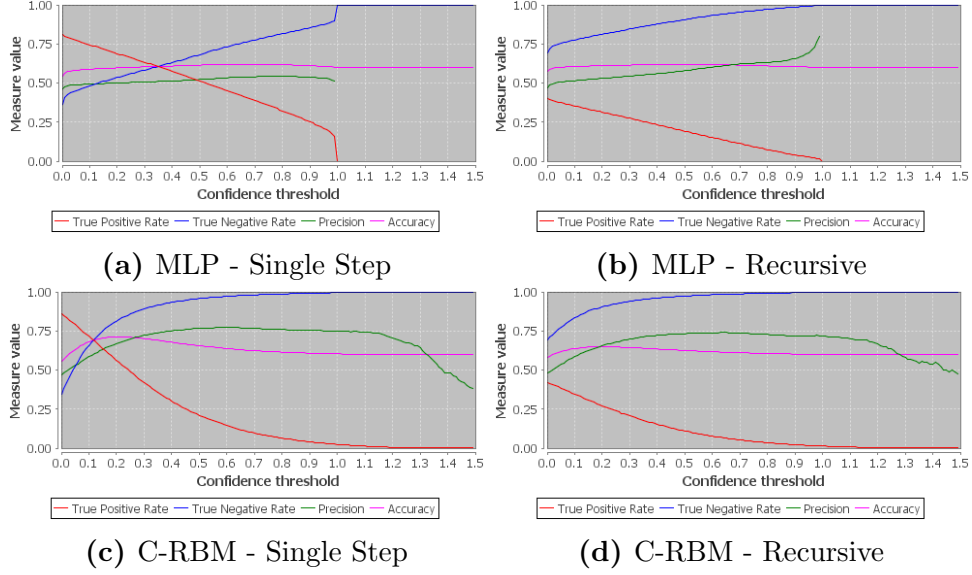


Figure C.2: Results of the experiments that used **2** historic time-steps to make predictions.

### C.2.2 3 historic time-steps

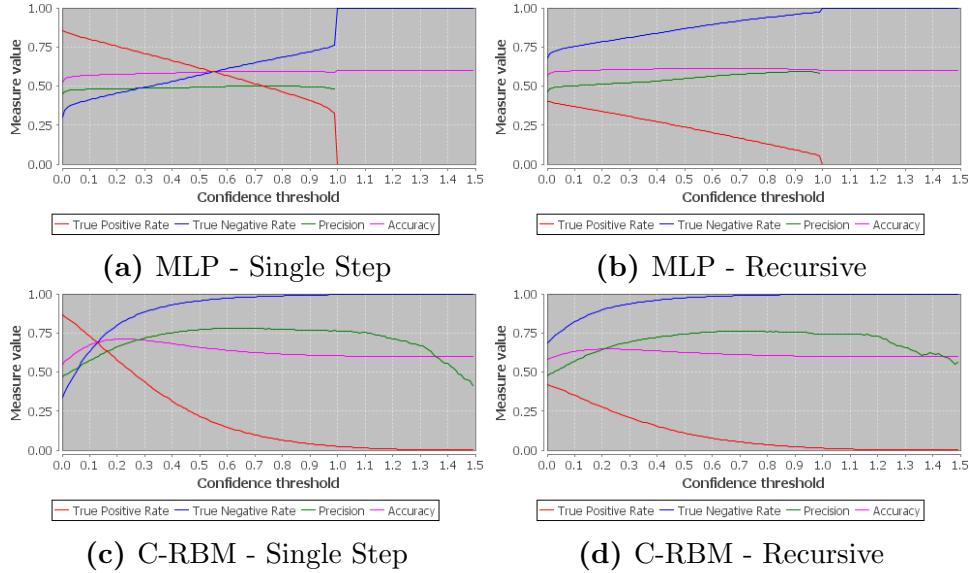


Figure C.3: Results of the experiments that used **3** historic time-steps to make predictions.

### C.2.3 5 historic time-steps

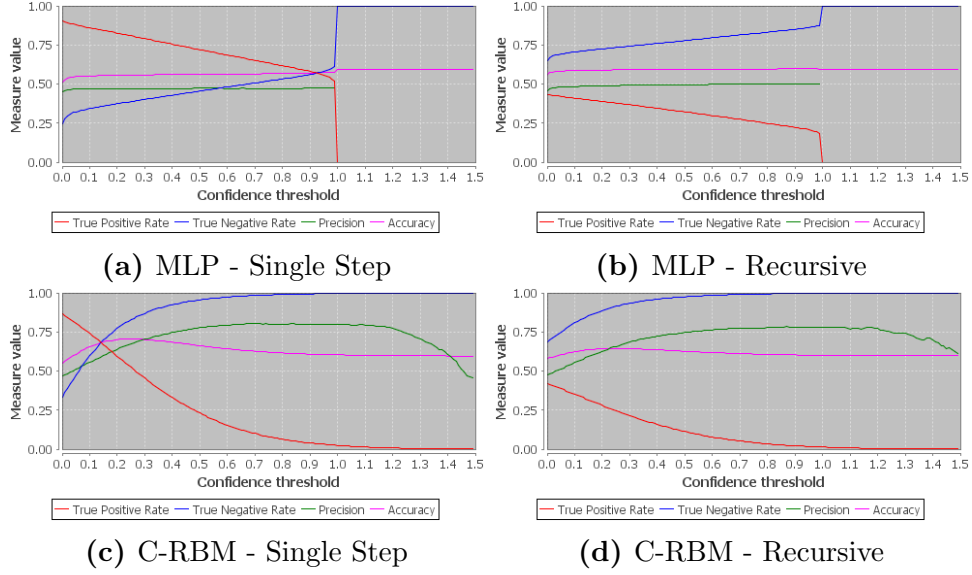


Figure C.4: Results of the experiments that used 5 historic time-steps to make predictions.

### C.2.4 6 historic time-steps

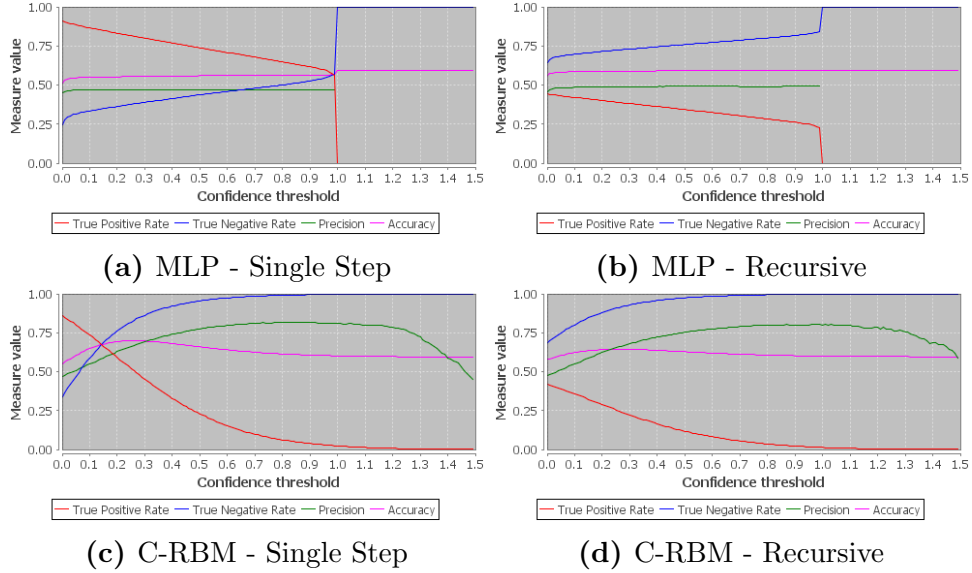


Figure C.5: Results of the experiments that used 6 historic time-steps to make predictions.

## C.3 Number of predicted time-steps

### C.3.1 3 predicted time-steps

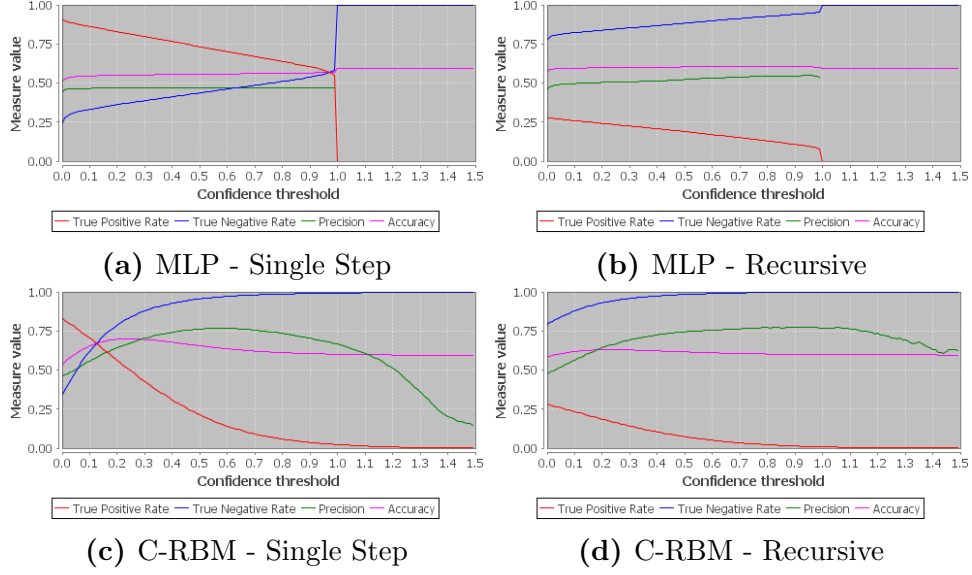


Figure C.6: Results of the experiments that predicted 3 time-steps.

### C.3.2 4 predicted time-steps

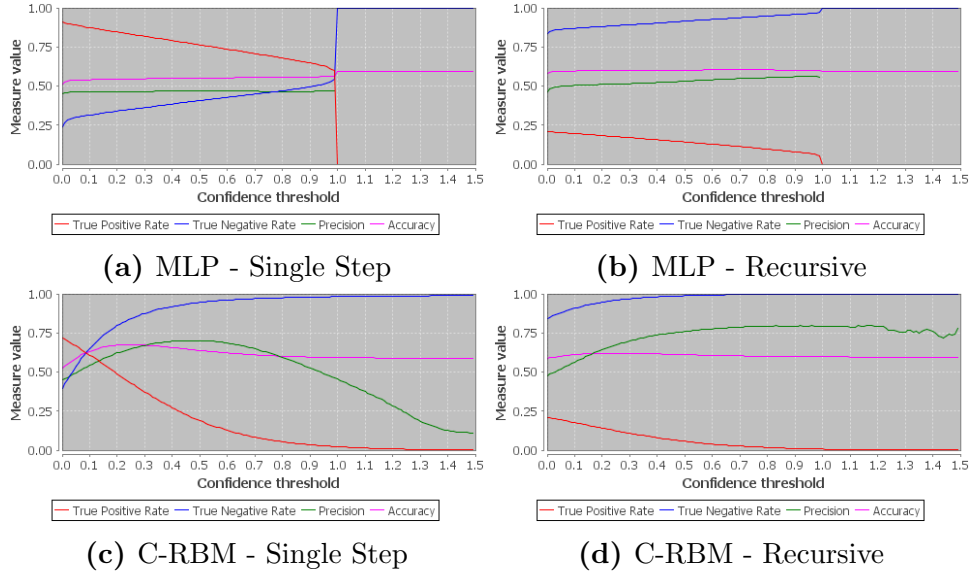
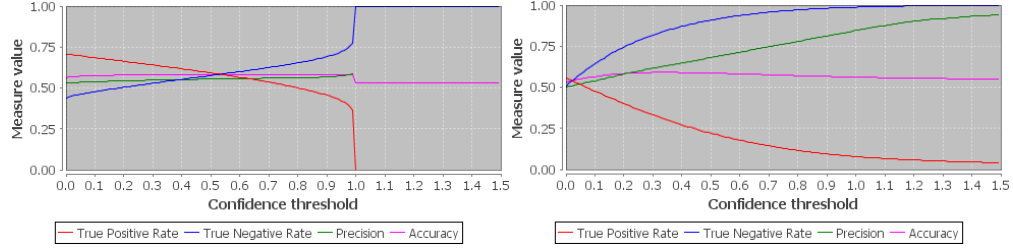


Figure C.7: Results of the experiments that predicted 4 time-steps.

## C.4 PCA features

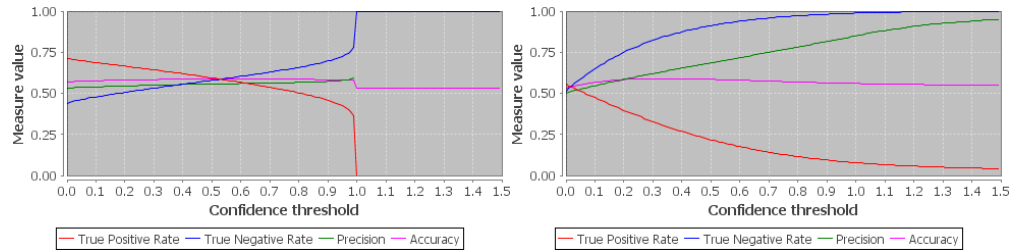
### C.4.1 PCA features covering 75% variance



(a) PCA 075% - MLP - Single Step (b) PCA 075% - C-RBM - Single Step

**Figure C.8:** Results of the experiments that used additional features, created with PCA, covering 0.75% variance of the data.

### C.4.2 PCA features covering 0.95% variance



(a) PCA 095% - MLP - Single Step (b) PCA 095% - C-RBM - Single Step

**Figure C.9:** Results of the experiments that used additional features, created with PCA, covering 0.95% variance of the data.

## C.5 Cap z-scores

### C.5.1 Capping z-scores at 1.75

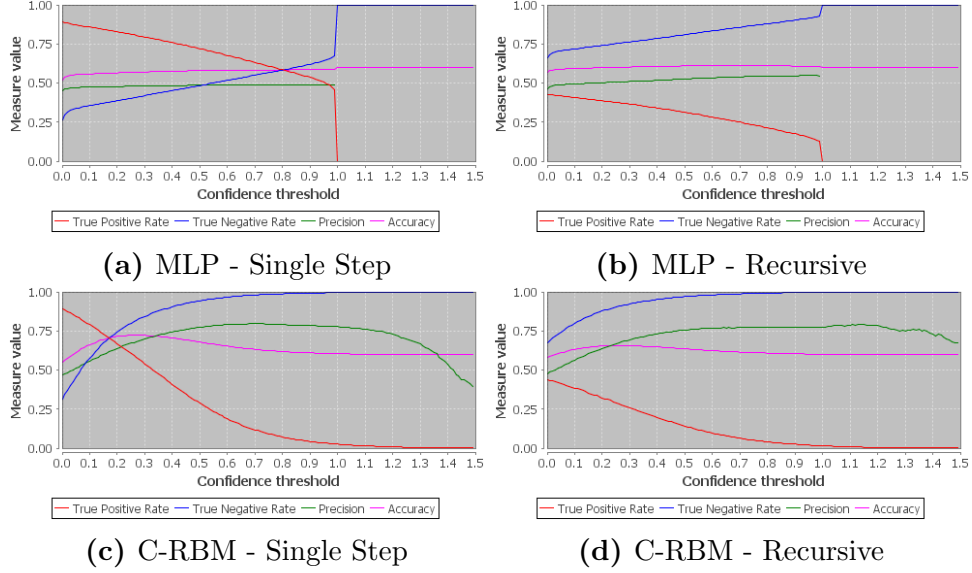


Figure C.10: Results of the experiments that capped the calculated z-scores at **1.75**.

### C.5.2 Capping z-scores at 2.00

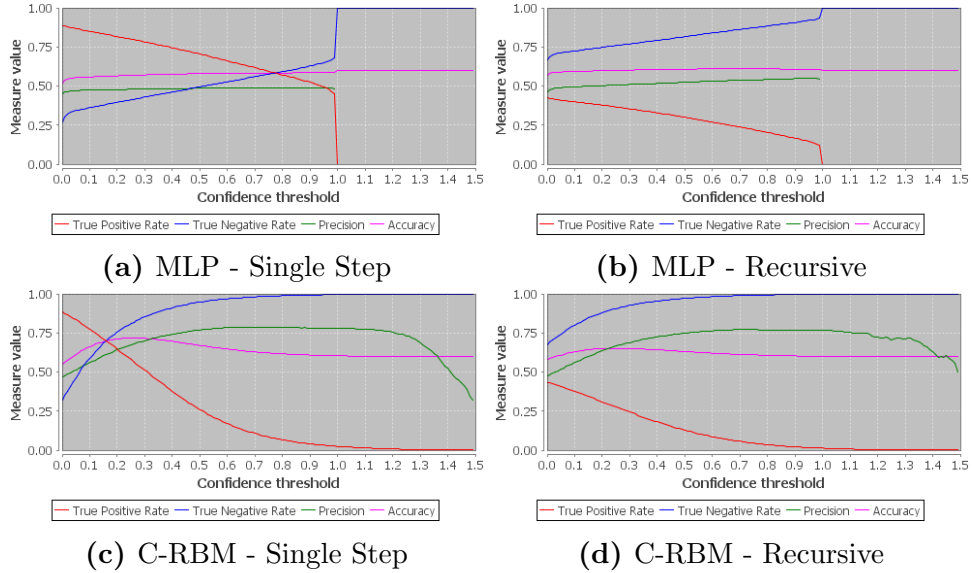


Figure C.11: Results of the experiments that capped the calculated z-scores at **2.00**.

### C.5.3 Capping z-scores at 2.25

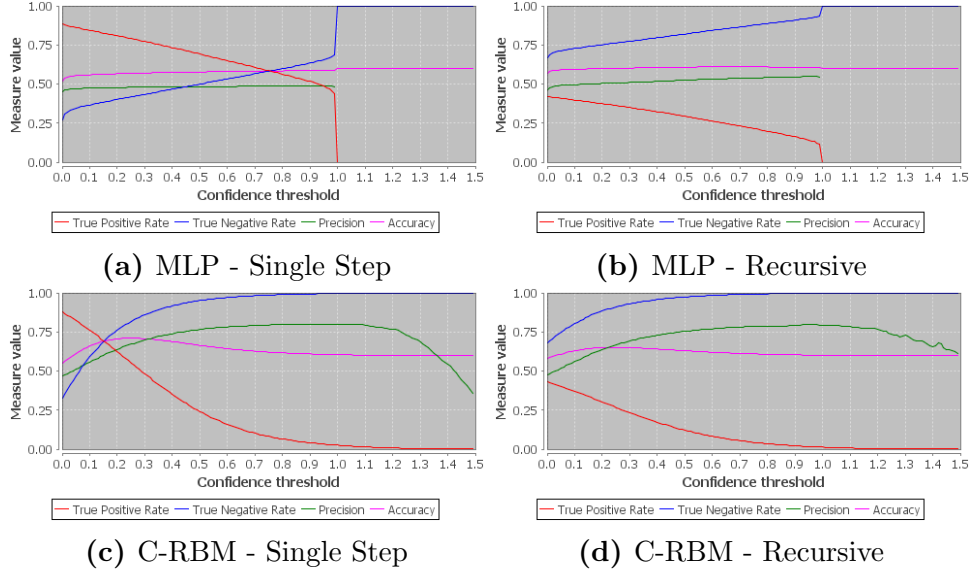


Figure C.12: Results of the experiments that capped the calculated z-scores at **2.25**

### C.5.4 Capping z-scores at 2.50

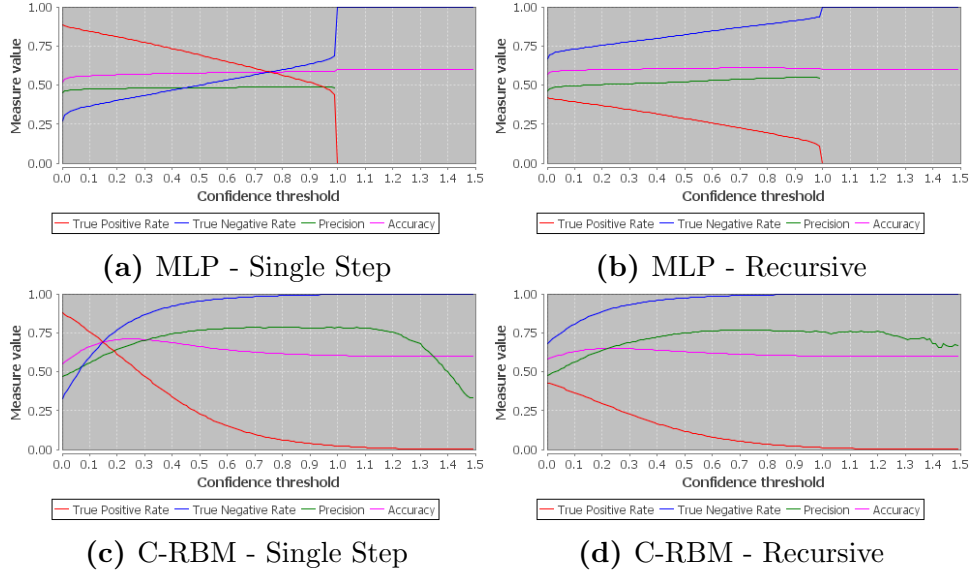


Figure C.13: Results of the experiments that capped the calculated z-scores at **2.50**.

## C.6 Learning rate

### C.6.1 Learning rate 0.01

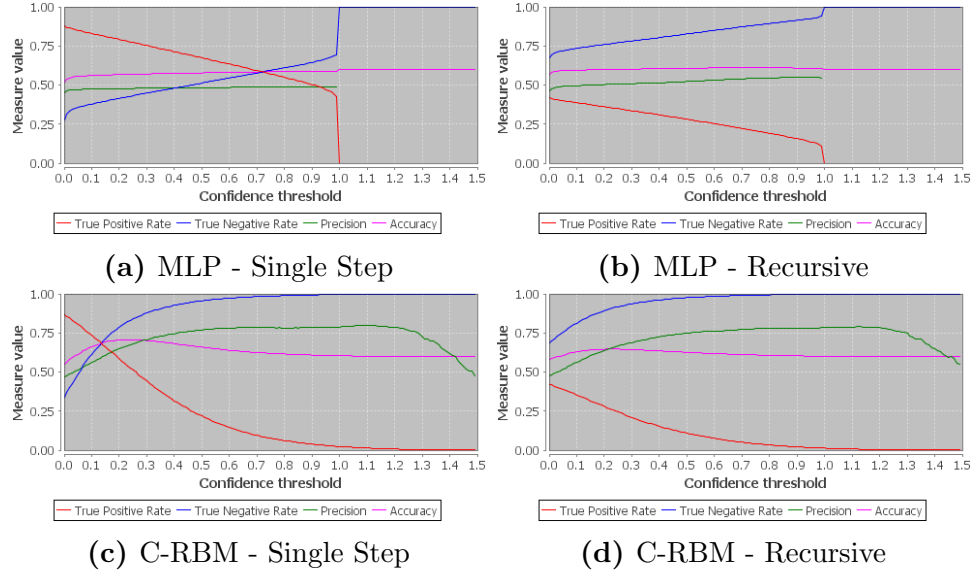


Figure C.14: Results of experiments utilising a learning rate of 0.01 for training.

### C.6.2 Learning rate 0.05

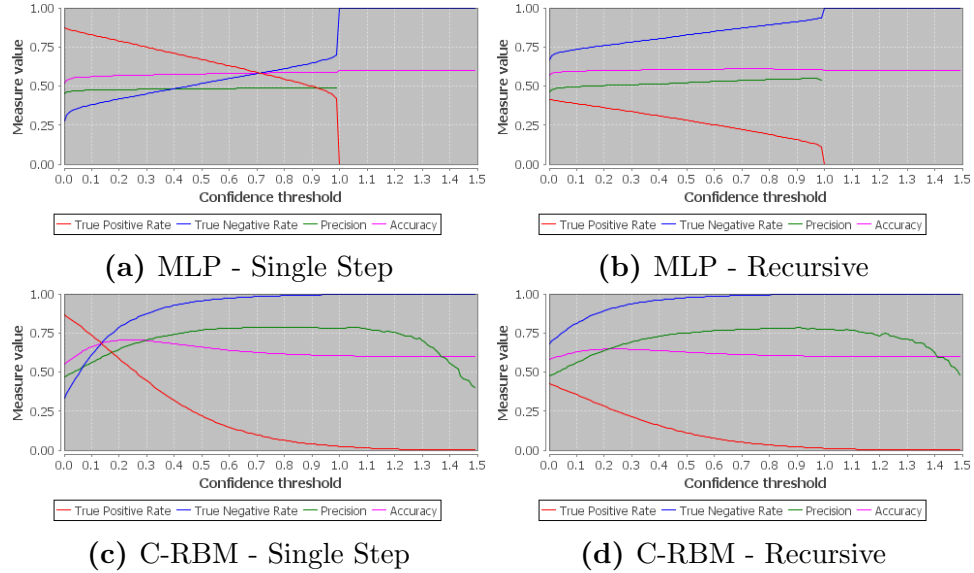


Figure C.15: Results of experiments utilising a learning rate of 0.05 for training.



## C.7 Epochs

### C.7.1 500 epochs

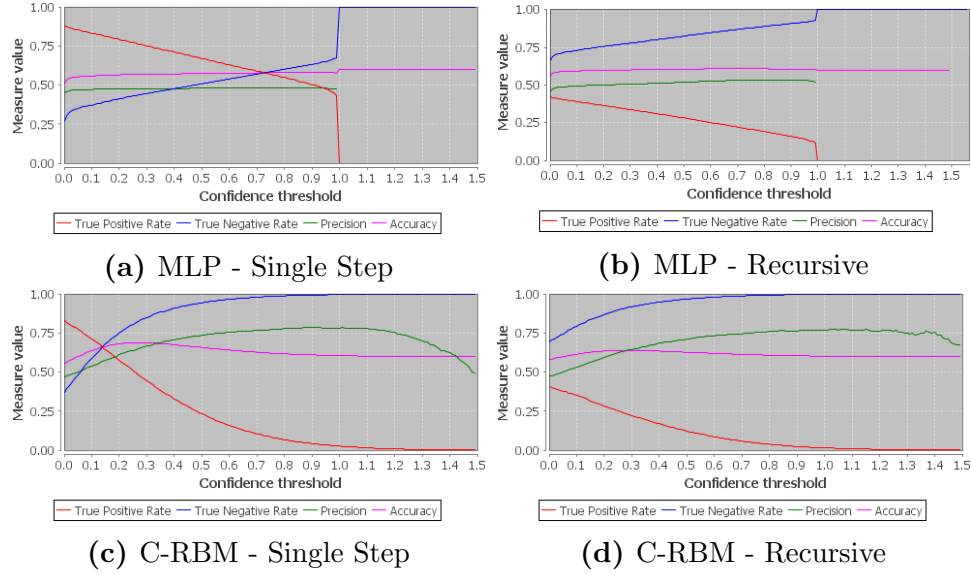


Figure C.16: Results of experiments that trained an algorithm for 500 epochs.

### C.7.2 2,500 epochs

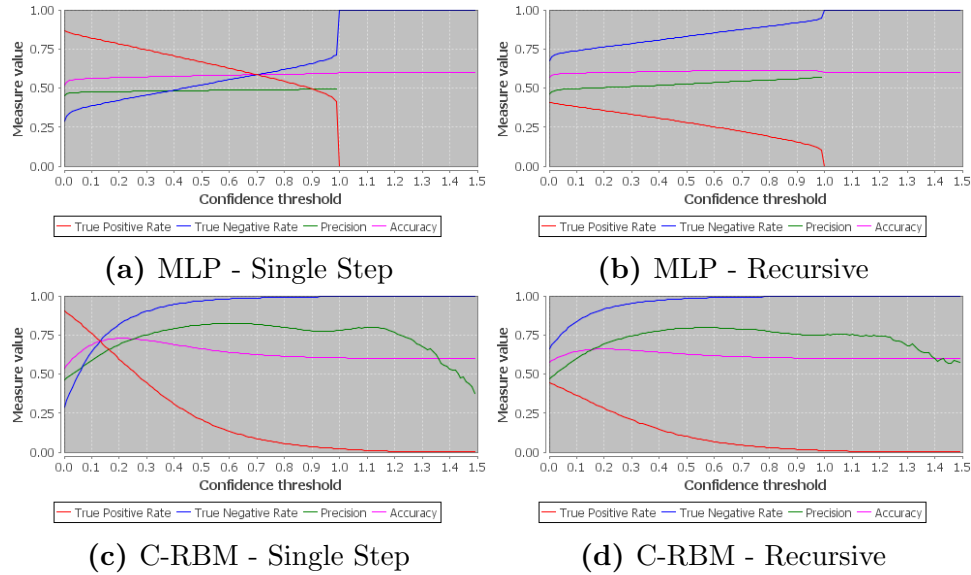
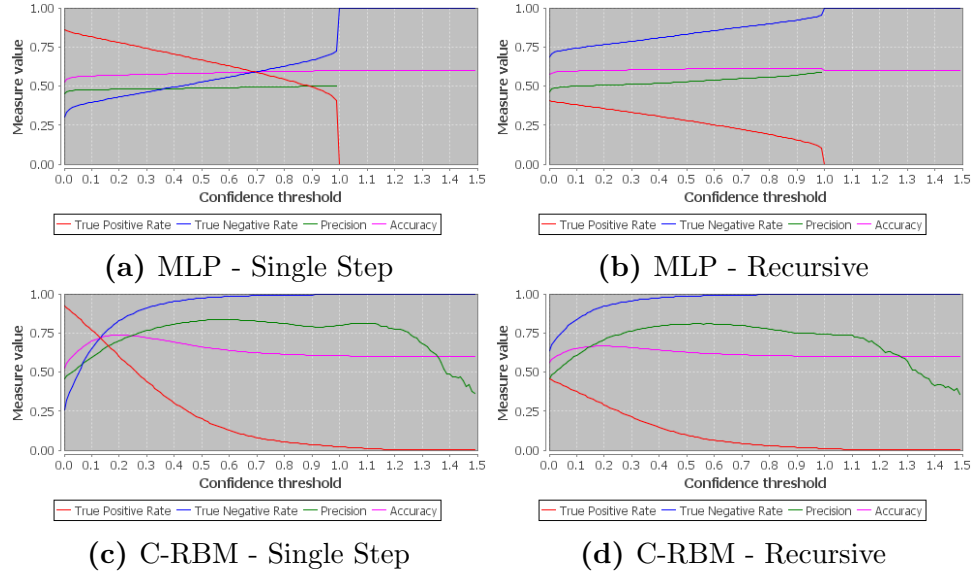


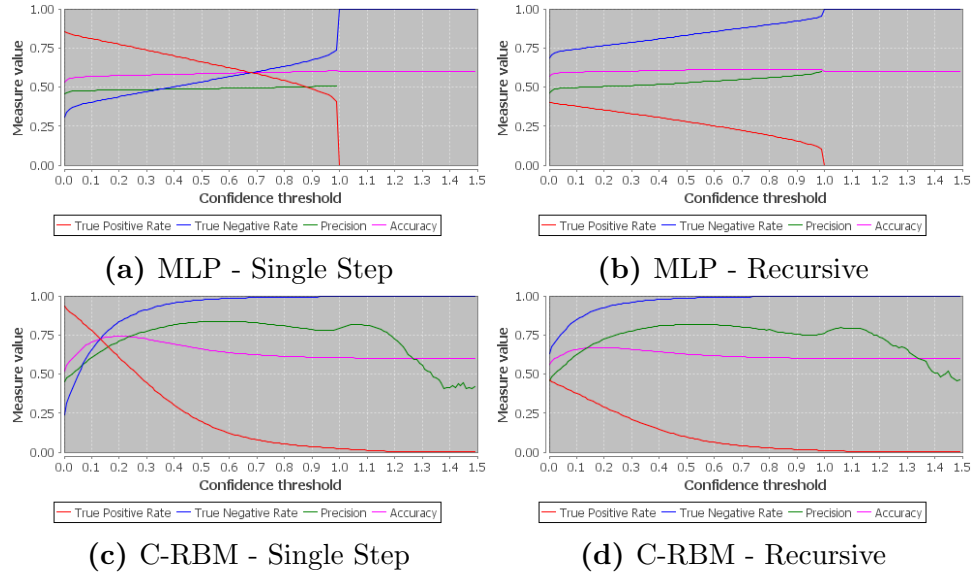
Figure C.17: Results of experiments that trained an algorithm for 2500 epochs.

### C.7.3 5,000 epochs



**Figure C.18:** Results of experiments that trained an algorithm for 5,000 epochs.

### C.7.4 10,000 epochs



**Figure C.19:** Results of experiments that trained an algorithm for 10,000 epochs.

## C.8 Number of hidden units

### C.8.1 0.33 hidden unit ratio

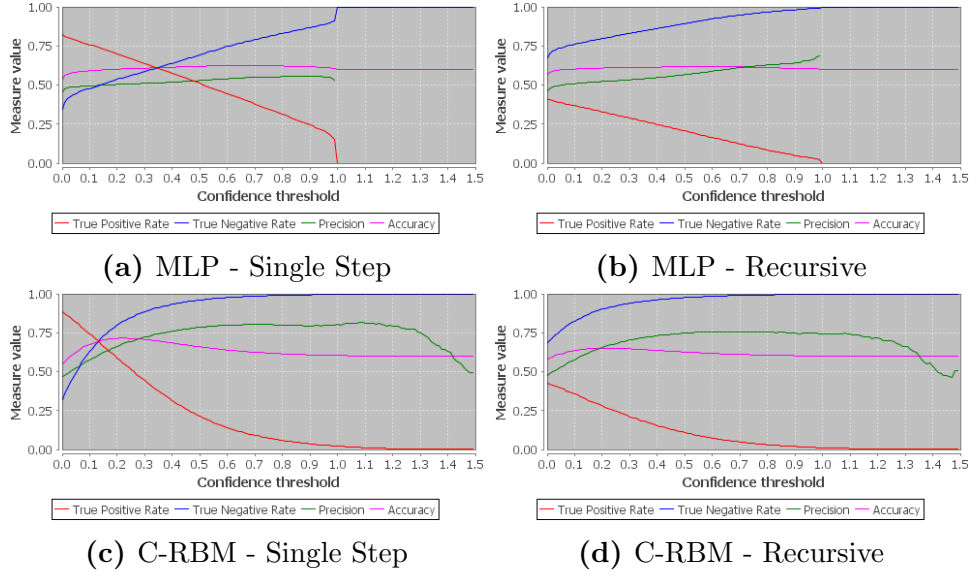


Figure C.20: Results of experiments with models utilising a hidden unit ratio of 0.33.

### C.8.2 0.66 hidden unit ratio

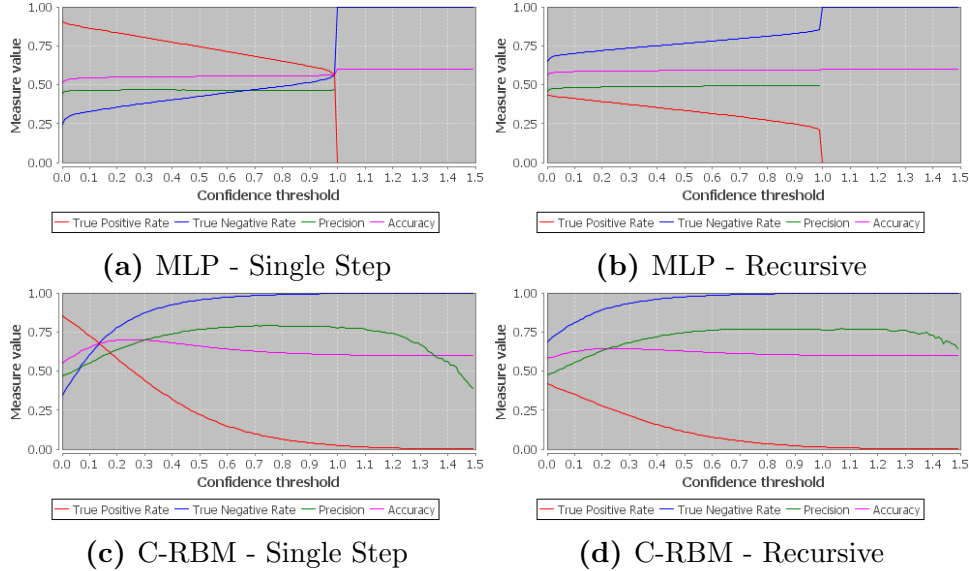


Figure C.21: Results of experiments with models utilising a hidden unit ratio of 0.66.