Summary

Soft-errors in an operating circuit are characterised by a bit-flip, the consequence of which varies in severity. The main natural source of these is cosmic rays which are highly charged particles originating from outer space. This makes it hard to eliminate soft-errors at their source. Furthermore, soft-errors are becoming an increasing concern with the constant demand and development of chips with increased density. This motivates the search for new and efficient soft-error mitigation techniques.

In the period between 2016 and 2018, several experiments were conducted where embedded computers were subjected to high levels of ionising radiation resembling the radiation caused by cosmic rays. During these experiments logs from the system operations were collected to supply data for later analysis and development of detection and resilience methods. One dataset created based on these experiments is the "LANSCE 18 Cruise" dataset. This dataset consists of logs from repeated simulations of a car driving with cruise control.

In 2022 the first study analysing the "LANSCE 18 Cruise" dataset was presented. Here different data representations to improve the preconditions for detecting soft-errors with machine learning were explored. In this study, we continue researching the "LANSCE 18 Cruise" dataset although with a focus on exploring different relevant machine-learning models. Note that we also utilise two extensions to the "LANSCE 18 Cruise" dataset.

Typically for this type of dataset, the data is naturally represented as a time-series. That is a collection of observations over time. However, we further experiment with the data transformation known as matrix profile. A matrix profile is a meta time-series which is developed to support time-series analysis tasks.

The approach of this study is to explore a variety of sequence-based machine-learning models as potential solutions to the binary classification problem of predicting whether or not a soft-error has occurred in a given log. The models in focus are: Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), Random Convolutional Kernel Transform (ROCKET), and HIVE-COTE 2.0 (HC2). Within the experimentation on the original data representation, ROCKET reached the highest accuracy of 0.67. However, transforming the data to matrix profiles was found to yield a general increase in accuracy. Here ROCKET reached an accuracy of 0.96 which is a percentage increase of about 43%. Furthermore, when transforming the data to matrix profiles, GRU was similarly found to reach an accuracy of 0.99 which is a percentage increase of about 94%. These results suggest that machine learning is a promising approach for detecting soft-error occurrences in PID-controlled environments. It further indicates that transforming the data to matrix profiles yields valuable information for the machine learning models.

Classifying Abnormal Behaviour Caused by Soft-errors in Logs From PID-controlled Environments Using Machine Learning Models

A Study on the "LANSCE 18 Cruise" Dataset

Master Thesis
Group cs-23-mi-10-14
Computer Science
Aalborg University
22. May 2023

AALBORG UNIVERSITY
STUDENT REPORT

Tenth Semester
Electronics and IT
Selma lagerløfs Vej 300
Aalborg University
9220 Aalborg

Project Title:

Classifying Abnormal Behaviour Caused by Soft-errors in Logs From PID-controlled Environments Using Machine Learning Models - A Study on the "LANSCE 18 Cruise" Dataset

Project Period:

Spring Semester 2023

Project Group:

cs-23-mi-10-14

Members:

Anna Veibel Bonde
Simon Kanne Mikkelsen

Supervisor:

Sean Kauffman
René Rydhof Hansen
Kim Guldstrand Larsen

Page Number Total: 87
Page Number Main: 44
Appendix: 33
Completion Date: 22-05-2023

Abstract:

As soft-errors become an increasing concern it motivates the search for new and efficient soft-error mitigation techniques. This study explores a variety of sequence-based machine-learning models as potential solutions to classifying occurrences of soft-errors. We hence examine the binary classification problem of predicting whether or not a soft-error has occurred in a given sample. The models in focus are: RNN, LSTM, GRU, ROCKET, and HIVE-COTE 2.0. These were trained on log data gathered from a PID-controlled system that was exposed to an accelerated neutron beam. This beam consequently caused soft-errors to happen within some of the logged data.
Through our experimentation, we reached an accuracy of 0.67 with ROCKET on the original data representation. However, transforming the data to matrix profiles was found to result in a general increase in accuracy. Here ROCKET reached an accuracy of 0.96. Furthermore, when transforming the data to matrix profiles GRU was similarly found to reach an accuracy of 0.99. These results suggest that machine learning is a promising approach for detecting soft-error occurrences in PID-controlled environments.

Preface

This master thesis was undertaken by the project group cs-23-mi-10-14 during the tenth-semester of computer science at Aalborg University, in spring 2023 under the guidance of Sean Kauffman, René Rydhof Hansen, and Kim Guldstrand Larsen.

Group cs-23-mi-10-14 - Computer Science - Aalborg University

Anna Veibel Bonde

Simon Kanne Mikkelsen

Contents

1 Introduction

In an operating circuit, a Single Event Error (SEE) is a disruption caused by an electrical disturbance. When discussing naturally occurring SEEs the electrical disturbance is typically caused by a single ion passing through or near a sensitive node in the circuit. The effect of the SEE may be destructive or non-destructive and are as such divided into two groups namely hard- and soft-errors, of which we focus on the latter. Soft-errors are characterised as a bit-flip within a memory cell caused by an electrical disturbance. The following consequences of which can vary in severity. The main source of soft-errors is cosmic rays from outer space which makes them hard to eliminate at their source. The rate at which these types of errors occur depends on different factors such as altitude and chip density. Soft-errors are of increasing concern due to the constant demand and development of chips with increased density. [1, 2]

The current methods used for soft-error detection and mitigation among others include Error correcting code (ECC) and replication at both the software and hardware level. All of these techniques add an extra layer of hardware requirements and complexity that with other techniques possibly could be solved more efficiently. [3]

In 2018 the "LANSCE 18 Cruise" dataset was created, where a system was subjected to a high level of ionising radiation resembling that caused by cosmic rays. This was achieved by exposing the system to an accelerated neutron beam. The dataset as such consists of system operation logs of which a subset registers occurrences of soft-errors. The purpose of creating this dataset was to provide real-world data for later analysis to develop detection and resilience methods.[4]

Mikkelsen and Bonde [3] presented in 2022 their results from an initial analysis and experimentation of the "LANSCE 18 Cruise" dataset. The goal of this study was to explore different data representations to improve the preconditions for detecting soft-errors with machine learning by experimenting with different sampling approaches.

In this study, we continue this research on the "LANSCE 18 Cruise" dataset. Here we shift the focus from a general analysis and understanding of the dataset to exploring different relevant machine-learning models.

Problem Analysis 2

This chapter will present further relevant domain knowledge to continue the research of Mikkelsen and Bonde [3]. This includes an introduction to the "LANSCE 18 Cruise" dataset and PID controllers. We furthermore present some related works among others in terms of machine-learning models that have shown potential in similar tasks.

2.1 The "LANSCE 18 Cruise" Dataset

Within this study, we employ the "LANSCE 18 Cruise" dataset[5]. It was created back in 2018 to capture real-world occurrences of soft-errors in system operation logs. The dataset was created by simulating a 2017 Toyota Corolla driving on flat ground and exposing the PID controller, representing the cruise control, to an accelerated neutron beam. [4, 5]

The dataset includes a total of 6 860 independent system log files from different experiment runs. These log files fall into two categories; nominal- and experiment-type. Nominal-type logs were created while the accelerated neutron beam was turned off and acted as the ground truth for the expected behaviour of the system given the experiment settings. Experiment-type logs were created while the beam was on, however, it is not guaranteed that these experience any abnormal behaviour. Abnormal behaviour within a log file is labelled with an off-nominal flag. This flag is set by comparing each experiment-type log with a nominal-type log from experiments with the same settings. Only 58 of the 6 725 experiment-type logs sets the off-nominal flag. Logs from experiments with the same settings and with no abnormal behaviour are equivalent which leaves the dataset with only 111 unique log files. Table 2.1 shows some relevant numbers in terms of the size of the dataset.

Type	Number of logs
Experiment	6 725
Nominal	135
Total	6 860

Table 2.1. Dataset numbers of the original LANSCE 18 Cruise dataset [5].

2.2 PID Controllers

PID controllers are widely used in the industry today as a mechanism for automatic feedback control. They work by applying corrections to a control function given the current values of the system. This is done by continuously computing an error value [1] and then applying a correction based on the three terms: proportional, integral and derivative. [6]

[1] The difference between a desired setpoint and a measured process variable

A typical example of a PID controller is the cruise control in a car. Here the measured process variable is the current speed of the car while the setpoint is the desired speed. When the environment changes eg. when the car reaches a hill, then the engine power needed to keep the desired speed is increased as a result. The PID controller then restores the speed with minimal delay by regulating the power from the engine in a controlled manner.

In summary, PID controllers regulate system outputs for different processes even though these may be disturbed, such as a car reaching a hill. However, if the integrity of the PID controller itself is compromised eg. by a soft-error, bit-injection, or if a sensor in the process is faulty and as a consequence feeds wrong information to the PID controller, this will cause a problem for the overall process. It would therefore be interesting to research whether we can detect faults in the system given the output from a PID controller or the process it is interacting with.

2.3 Related Work

Machine learning has shown great potential in domains such as maintenance and anomaly detection. Machine learning models can learn complex patterns and behaviours that can be used to efficiently solve tasks that would otherwise be difficult and computationally complex to solve. [7, 8]

One category of machine learning models which we focus on in this study is sequence-based models. These types of models train on sequential data such as time-series and text streams. A time-series is a collection of observations over time that can be used to analyse trends and patterns. The logs from the "LANSCE-18 Cruise" dataset can be represented as time-series as it is comprised of data logged over time. As we represent the dataset as such, the data transformation known as matrix profile would be worth investigating.

In regards to the sequence-based machine learning models, two subcategories are of interest in this study namely the Recurrent Neural Network (RNN) and time-series classification models. We will in the following sub-sections introduce relevant machine learning models within these two sub-categories as well as matrix profiles. A more detailed description of all of these can be found in Chapter 4.

2.3.1 Matrix Profile

Yeh et al. [9] presented in 2016 the data transformation known as Matrix Profile which can be used to support different time-series analysis tasks. Two fundamental tasks are the identification of anomalies and trends. Given this research Benschoten et al. [10] presented in 2020 an API which leverages matrix profiles. Hence, this API supplies an array of functions which can support different time-series analysis tasks utilising matrix profiles.

2.3.2 Recurrent Neural Network

RNN is a type of Neural Network (NN) that handles sequential data and has the notion of memory. Hence, what previously appeared in a sequence affects the evaluation of the current step in the sequence. The simple RNN model often suffers from the exploding or vanishing gradient problem resulting in suboptimal learning. [11]

Hochreiter and Schmidhuber [12] presented in 1997 a model called Long Short-Term Memory (LSTM) that mitigates this exploding/vanishing gradient problem by considering both long- and short-term memory. It achieves this by introducing a range of gates that helps to forget and retain certain information when passing the output to the next step in the sequence. The LSTM model has gained widespread popularity and has for example been used by Google and Facebook to improve their services [13].

Chung et al. [14] introduced in 2014 a model called Gated Recurrent Unit (GRU) that is similar to LSTM in that they both retain long-term memory. GRU differentiates itself by having a simpler architecture and is, by extension, less computationally expensive. This makes it an easier model to train. However, on more complex tasks, the LSTM model still outperforms the GRU model.

2.3.3 Time-series Classification

Time-series classification is another subcategory of sequence-based machine learning models. These models are eg. used to identify unusual and anomalous behaviour within a time-series[15]. Two state-of-the-art time-series classification models [16] are Random Convolutional Kernel Transform (ROCKET)[17] and HIVE-COTE 2.0 (HC2)[18].

Dempster et al. [17] presented in 2020 a fast and accurate time-series classification model called ROCKET. It works by initialising a large number of random convolutional kernels that are used to transform the data into a linear classification problem. This is in part what makes ROCKET appealing as it can achieve similar results to other state-of-the-art models in a fraction of the time.

Bagnall et al. [18] introduced in 2021 the second major iteration of their HIVE-COTE time-series classification ensemble model. HC2 combines classification models from different domains. The previously described ROCKET[17] model is as of the second major iteration also part of this model.This ensemble model greatly outperforms other state-of-the-art time-series classification models when testing the performance on the UEA archive datasets[19]. The UEA archive contains a collection of a broad range of time-series classification datasets.

2.4 Problem Statement

Soft-errors are becoming an increasing concern as chip density grows. Furthermore, current mitigation techniques require additional or specialised hardware. This motivates the search for new and more efficient approaches for mitigating soft-errors. Machine learning is a natural area to explore as it has shown to have potential in similar complex tasks. The "LANSCE 18 Cruise" dataset contains data that can be used to train a machine learning model to classify soft-error occurrences and by extension, help to mitigate these in PID-controlled environments. We have, given these considerations, formulated the following problem statement:

Given logs from a PID-controlled system, to what extent can a machine-learning classifier detect the effect that soft-errors have on a system?

Datasets 3

We will, in this chapter, present the datasets used in this study to get an understanding of the data used when training the machine learning models in our experiments. These datasets include the "LANSCE 18 Cruise" dataset[5] introduced in Section 2.1 as well as two simulated extensions that were created at a later date. The chapter is based on: the "LANSCE 18 Cruise" dataset documentation[4], the findings of Mikkelsen and Bonde [3], and our own analysis.

3.1 Dataset Creations

This section will provide a brief summary of the creation process of the three datasets used. These are presented in order of their creation.

3.1.1 LANSCE 18 Cruise

The "LANSCE 18 Cruise" dataset[5] was created in 2018 at Los Alamos Neutron Science Center (LANSCE), New Mexico, USA. As presented in Section 2.1, the dataset was created by simulating a 2017 Toyota Corolla driving on flat ground. In the experimental setup used to construct the dataset, two Arduino Uno boards were used. Here one ran the PID controller, representing the cruise control of the car, while the other acted as the plant, representing the actual car with simple physics modelling. The Arduino Uno board running the PID controller was subjected to ionising radiation from an accelerated neutron beam. During the experiments, simulating the car driving, the CAN data of the two boards were captured. This information was then used to create the log files which constitute the "LANSCE 18 Cruise" dataset.

Note that the log files of this dataset are divided into two types, as presented in Section 2.1. The files of the first type, referred to as 'nominal', are from experiments conducted with the accelerated neutron beam turned off. On the other hand, the files of the second type, referred to as 'experiment', are from experiments conducted with the accelerated neutron beam turned on.

3.1.2 Nominal Extension

Section 2.1 remarks that the "LANSCE 18 Cruise" dataset in fact only contains 111 unique log files. To provide a larger dataset of unique log files the nominal extension was created. This extension was first presented in [3]. It was constructed by simulating the creation process of the "LANSCE 18 Cruise" dataset (Section 3.1.1). This extension only contains log files that all would have been considered nominal-type in the original dataset.

3.1.3 Fail-injected Extension

During this project, we gained access to another newly created extension to the "LANSCE 18 Cruise" dataset. The goal of this extension was to supply a number of log files that shows abnormal behaviour. As described in Section 3.2.2 this means that the log files at some point deviate in the high word, recorded from the CAN bus, compared to a nominal-type log with the same settings.

This dataset was created by simulating the creation process of the "LANSCE 18 Cruise" dataset, similar to the nominal extension. The difference is that soft-errors were simulated while creating the dataset. This was done by using fail-injection in the form of bit-flips. Some concepts and numbers presented in Section 3.2 are used to introduce the intuition behind the fail-injection implementation.

The two main aspects of the fail-injection to consider for understanding the implementation are where and when the bit-flips occur.

Firstly, during the creation of the "LANSCE 18 Cruise" dataset, the sum of a byte array[1] from the program memory was used in the computations performed by the PID controller. Simulating a soft-error was thus done by flipping one bit in this byte array. The bit to be flipped was chosen by using a uniform distribution. Consequently, when the bit is flipped the sum of the byte array would change from 0 to a power of 2.

The second aspect to consider is when the bit-flip is injected. These injections happen when the controller is performing its computations. Thus the abnormal behaviour in the log appears when the controller sends information about the torque to the plant. This will happen 1 500 times in each log. One of these 1 500 steps is sampled from a truncated normal distribution with a mean of 730 and a standard deviation of 447. The lower bound of this distribution is 0 while the upper bound is 1 499. This normal distribution is found by analysing the logs from the "LANSCE 18 Cruise" dataset that experience abnormal behaviour and at what time step this happens.

3.2 Structure of the Datasets

The three datasets presented in section 3.1 have a similar structure as the two extensions simulate the "LANSCE 18 Cruise" dataset creation process. Therefore, unless otherwise specified, it can be assumed that all information in the following section pertains to all three datasets.

3.2.1 Naming Scheme of Dataset Files

The datasets consists of a number of logs. These are identified by a unique run ID, which is further used as a naming scheme to save each log into individual CSV files. The run IDs consist of six parts: four experiment configurations, the type, and the number of the run. The four experiment configurations, henceforth referred to as settings, are those used as input to the experimental setup of the "LANSCE 18 Cruise" dataset creation. The types used are: 'nominal', 'experiment', and 'simulation'. Here, as already stated in section 2.1 and section 3.1.1, the logs of the "LANSCE 18 Cruise" dataset are either of the nominal-

[1] Initialised as a zero byte array

or experiment-type. On the other hand, the logs from the two extensions are all of the simulation-type.

3.2.2 Log Structure

The log of each run is saved as a table in CSV files. This table, as can be seen in Table 3.1, consists of seven columns namely: *timestamp*, CAN ID, *high word*, *low word*, *mem. sum* and *off nom.*. We will here present the meaning of the columns relevant to this study.

CAN ID is the descriptive name associated with the ID field of the logged CAN message. We delve deeper into these CAN IDs later in this section. The *high word* column contains the first 32-bits of the CAN-data, from the logged CAN message. The values are saved as signed integers. The *mem. sum* column is comprised of the sum of a byte array in the program memory at the time the corresponding CAN message was logged. This byte array was used to increase the sensitivity of the system to soft-errors. Finally, the *off nom.* column shows the status of the 'off nominal' flag. This flag is used to indicate if a run deviates in behaviour from a nominal-type run with the same settings. Thus, '0' means that the run has not yet deviated. When it is discovered that the log deviates from the expected behaviour the flag is raised and stays as such for the rest of the log.

Timestamp	CAN ID	HW	LW	flux	Mem. sum	Off nom.
1 538 668 440.030 839	PLANT_INFO_SimulationTime	0	0	368 372	0	0
1 538 668 440.031 831	PLANT_INFO_VehicleSpeed	361	0	368 372	0	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1 538 668 496.918 876	CTRL_INFO_TorqueCommand	64 661	0	526 246	4	1
1 538 668 496.929 3	CTRL_INFO_VehSpd_ECHO	1 500	0	526 246	4	1

Table 3.1. The first and last entries of the "kp122ki13kd1sp13experiment29" log. Note that *high word* and *low word* have been abbreviated to HW and LW respectively.

There are four types of CAN IDs present in the datasets, all of which are shown in Table 3.2. Together these four can be considered a step within a given log. Hence, the logs consist of these CAN IDs in a repeated pattern, or equivalently they are comprised of multiple steps.

The CAN IDs 'PLANT_INFO_SimulationTime', 'PLANT_INFO_VehicleSpeed', and 'CTRL_INFO_TorqueCommand' are the ones used when determining whether or not to raise the off-nominal flag. The flag is as such raised when the high word of these CAN messages differs from a corresponding nominal-type log.

Hex ID	Dec ID	Descriptive name in DB	Sender	Purpose	High word meaning	Low word meaning
130	304	PLANT_INFO_SimulationTime	Plant	Report the simulation time steps, beginning at zero	Simulation time in "steps"	NA
132	306	PLANT_INFO_VehicleSpeed	Plant	Report the current speed from the Plant (vehicle)	Current velocity	NA
210	528	CTRL_INFO_TorqueCommand	Controller	The controller command to set the torque	Calculated torque value	NA
211	529	CTRL_INFO_VehSpd_ECHO	Controller	Echo back the speed used in the torque calculation	Current velocity	NA

Table 3.2. The CAN IDs present in the dataset and relevant information pertaining to these [4].

3.2.3 Central Numbers

The numbers presented in this section exclude the three outliers[2] that deviate from the normal length of a log file. Further details on these three outliers can be found in Mikkelsen and Bonde [3].

The "LANSCE 18 Cruise" dataset contains 6 857 logs where 134 are of the nominal-type and the rest are of the experiment-type. However, as presented in Section 2.1 the "LANSCE 18 Cruise" dataset only contains 111 unique logs due to repeated execution of the same settings. Excluding the three outliers results in 108 unique runs. Of these unique logs, 52 are of the nominal-type and the other 56 are of the experiment-type. Note that given the way we extract the unique samples all 56 logs of experiment-type raises the off-nominal flag. A full list of the run IDs of these 56 runs can be found in Appendix B. The nominal extension dataset contains 794 630 logs that can be considered nominal-type. Additionally, the fail-injected extension dataset on the other hand contains 100 000 logs that all sets the off-nominal flag. Finally, Common for all three datasets is that each log consists of 1 500 steps (Section 3.2.2) which translates to 6 000 log entries.

[2] 'kp147ki5kd1sp7experiment57', 'kp274ki9kd3sp15experiment77', and 'kp695ki5kd2sp13experiment65'

Background 4

In this chapter, we present the models used in our experimentation in a more formalised manner. We first define a time-series as this is how we represent the data. Matrix profiles are also defined as we use this data transformation to highlight anomalies in the data. We then examine the different machine learning models used in this study, which are divided into baseline-, RNN-, and time-series classifications models. Finally, we present the metrics used for evaluating the performance of the machine learning models.

4.1 Time-series

As presented in Section 2.3, a time-series is a collection of observations gathered over time. These observations can then be used to analyse trends and patterns.

Definition 4.1.1. *A* time-series $\mathbf{T}$ *is a sequence of* n *real-valued vectors containing* d *observations each. That is* $\mathbf{T} = \mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_n$ *where* $\mathbf{t}_i \in \mathbb{R}^d$ *for* $1 \leq i \leq n$.

From Definition 4.1.1 we see how each time step $\mathbf{t}_i$ contains d observations. An example of these observations, related to the datasets used in this study, could be: Given a log from the datasets, a time-series with 1 500 time-steps (Section 3.2.3) where each time-step is a two-dimensional vector can be generated. Let $\mathbf{T}$ be such a time-series. Then each step contains two observations such that $\mathbf{t_i} = \langle \theta_i, \psi_i \rangle$ for $1 \leq i \leq 1500$. Here θ_i and ψ_i are the high words of the log entries, at time-step i, with the CAN IDs "CTRL_INFO_-TorqueCommand" and "PLANT_INFO_VehicleSpeed" respectively.

If the value d in Definition 4.1.1 is equal to one, the time-series is univariate. Whereas if d is greater than one, the time-series is multivariate.

We define a subsequence of a time-series as seen in Definition 4.1.2. Note that a subsequence given this definition is in itself also a time-series.

Definition 4.1.2. *Given a time-series* $\mathbf{T}$ *with length* n*, a* subsequence $\mathbf{S}_{i,L}$ *of* $\mathbf{T}$ *is a sampling of length* $L \leq n$ *of contiguous position from* i*. That is* $\mathbf{S}_{i,L} = \mathbf{t}_i, \mathbf{t}_{i+1}, \ldots, \mathbf{t}_{i+(L-1)}$ *for* $1 \leq i \leq n - L + 1$.

4.2 Matrix Profile

When analysing time-series data, one is usually interested in identifying trends and anomalies [20]. A trend within a time-series is a repeated subsequence in regard to its visual "shape", whereas an anomaly is a subsequence that is unique from the rest of the

time-series. In our case, we want to identify if an anomaly has happened within a log in the dataset. A data structure developed to ease the process of finding anomalies in a time-series is matrix profiles[10]. A matrix profile is intuitively a collection of comparisons between all subsequences of a specific length. It stores the difference between a subsequence and its most similar counterpart among all other subsequences of the same length. If a sub-sequence is part of a trend the difference stored by the matrix profile would be zero, or close to zero if the data contains noise. On the other hand, as anomalies usually look unique they lead to high values in the matrix profile, also called discords. In Figure 4.1, we see an example, from the "Lance 18 Cruise" dataset, of how the anomaly due to its unique "shape" leads to high values in the matrix profile compared to the rest of the profile.

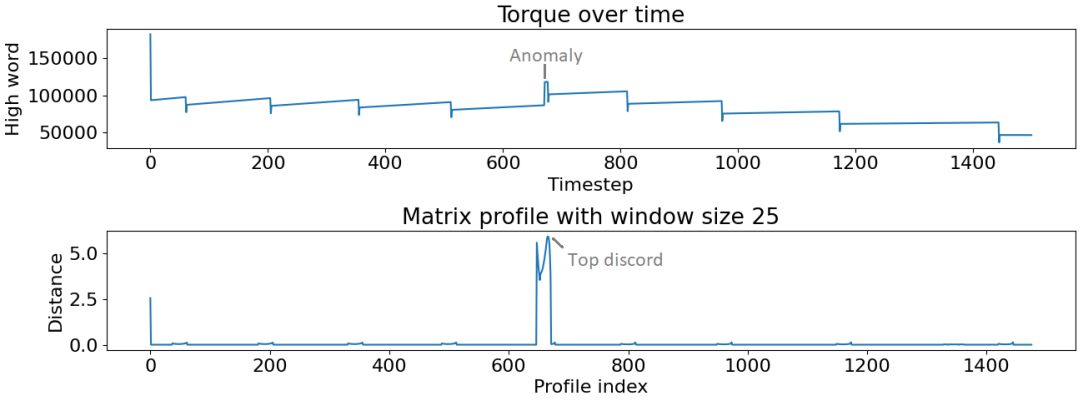


Figure 4.1. (Top) The high word values of the 'CTRL_INFO_TorqueCommand' from the log 'kp105ki8kd1sp20experiment54' plotted as a time-series, with the abnormal behaviour marked. (Bottom) The matrix profile of the time-series from the above plot, with the top discord marked.

When formally defining a matrix profile we first need to understand the distance function used to compare two time-series. To determine the distance between two time-series $\mathbf{T}$ and $\mathbf{S}$ of the same length n, the first step is to individually z-normalise them. That is where each time-series is normalised to have a mean of zero and a standard deviation of one. This is done to not let the distance measure be affected by the off-sets of the two time-series and rather let it be an evaluation of their respective "shapes". Once the two time-series have been z-normalised the Euclidean distance function is applied as can be seen in Equation 4.1.[21, 22]

$$dist(\mathbf{T}, \mathbf{S}) = \sqrt{\sum_{i=1}^{n}(t_i - s_i)^2} \tag{4.1}$$

As noted in Section 4.1 a subsequence of a time-series is also a time-series. Thus we can use the approach just presented to compute the distance between two subsequences of the same length as well. This leads to the definition of a matrix profile below.[22, 23]

Definition 4.2.1. *Given a univariate time-series* $\mathbf{T}$ *with length* n*, the* matrix profile $\mathbf{P}_L \in \mathbb{R}^{n-L+1}$ *of* $\mathbf{T}$ *is a meta time-series that consists of the shortest z-normalised Euclidean distance of each subsequence of length* L *and all other subsequences of the same length. That is* $\mathbf{P}_L = p_1, p_2, \ldots p_{n-L+1}$ *s.t.* $p_i = min\{dist(\mathbf{S}_{i,L}, \mathbf{S}_{j,L}) | \forall j \in \{1, 2, \ldots, n-L+1\} \wedge i \neq j\}$

Remark When talking about matrix profiles the term *window size* is used interchangeably with the considered subsequence length L.

Remark As subsequences of a time-series $\mathbf{T}$ that are close to each other in regards to starting position have a tendency to be similar in distance, an exclusion zone is applied when computing the matrix profile. Applying the exclusion zone to the matrix profile has the following effect: Subsequences, where the starting point is in the exclusion zone, are disregarded when determining the shortest distance to other subsequences. The exclusion zone is set to be the area of half the window size, L, both before and after the start point, i of the subsequence that is considered $\mathbf{S}_{i,L}$

4.3 Baseline Models

In this section, we present the models Logistic Regression (LR) and Support Vector Machine (SVM) based on [24, 25]. We refer to these as baseline models as the results of experiments run with these models (Section 6.3) are used as a baseline for the general experimentation.

Before presenting these two models we introduce a bit of notation. The goal of the machine learning models is to predict the class of the time-series given as input. The class of a time-series is set to one if abnormal behaviour caused by soft-errors is present and zero otherwise. We differentiate between the actual class label of a time-series $y \in \{0, 1\}$ and the predicted class $\hat{y} \in \{0, 1\}$. Finally, the dataset of m univariate time-series is denoted as $\mathcal{X} = \{\mathbf{T}_i | \forall i \in \{1, 2, \ldots, m\}\}$ where all time-series are of the same length n.

4.3.1 Logistic Regression

Once the LR model is fitted we predict the target value $\hat{y}_{new}$ of a new sample $\mathbf{T}_{new}$ by first computing the probability of that sample experiencing an anomaly. That is $P(y_{new} = 1 | \mathbf{T}_{new})$. This probability is computed using the function $\hat{p}$ from Equation 4.2. Here $\mathbf{w} \in \mathbb{R}^n$ is the weight vector and b is the bias term. Note that the dimensionality of the weight vector and the length of $\mathbf{T}_{new}$ must be equal to the length of the time-series within the dataset $\mathcal{X}$. After computing $\hat{p}$, a threshold of 0.5 is used to determine the predicted class as seen in Equation 4.2.

$$\hat{y}_{new} = \begin{cases} 1, & \hat{p}(\mathbf{T}_{new}) \geq 0.5 \\ 0, & \text{otherwise} \end{cases} \qquad \text{where} \qquad \hat{p}(\mathbf{T}_{new}) = \frac{1}{1 + e^{(-\mathbf{T}_{new}\mathbf{w} - b)}} \tag{4.2}$$

Given a dataset $\mathcal{X}$ of m samples we fit the model by minimising the following cost function (Equation 4.3) in regards to the weights and the bias term of the prediction function $\hat{p}$. These updated weights and bias term are then used in Equation 4.2 when predicting the target values of the given samples.

$$\mathbf{w}, b = \arg\min_{\mathbf{w}, b} \sum_{i=1}^{m} (-y_i log(\hat{p}(\mathbf{T}_i)) - (1 - y_i) log(1 - \hat{p}(\mathbf{T}_i))) + \frac{1}{2} ||\tilde{\mathbf{w}}||_2^2 \tag{4.3}$$

In Equation 4.3 $\tilde{\mathbf{w}} \in \mathbb{R}^{n+1}$ is the vector resulting from prepending the bias term to the weight vector such that $\tilde{\mathbf{w}} = [b\mathbf{w}] = \langle b, w_1, \ldots, w_n \rangle$.

4.3.2 Support Vector Machine

For a binary classification task, the SVM optimises the decision boundary given by Equation 4.4 by maximising the margin. The margin is the smallest distance between the decision boundary and any samples used when optimising the model. If the dataset is non-linearly separable the samples are mapped to a higher-dimensional feature space where they become linearly separable.

$$db(\mathbf{T}) = \mathbf{w}\phi(\mathbf{T}) + b \tag{4.4}$$

In Equation 4.4, $\phi : \mathbb{R}^n \to \mathbb{R}^D$ is a fixed mapping function which maps $\mathbf{T}$ of length n to a feature space where the data is linearly separable. It is typically the case for this mapping that $n < D$. Furthermore, $\mathbf{w} \in \mathbb{R}^D$ is the weight vector of the constructed decision boundary and b is the bias term.

Given a dataset $\mathcal{X}$ of m samples and the vector $\mathbf{y} \in \mathbb{R}^m$ where y_i is the class label of sample $\mathbf{T}_i$ for all $i = 1, \ldots, m$ we optimize the db function using Equation 4.5. With this optimization problem, we try to maximise the margin by minimising $||\mathbf{w}||^2$. As the dataset may not necessarily be perfectly separable ζ_i is added to the optimization problem. ζ_i is a distance measure we allow sample $\mathbf{T}_i$ for $i = 1, \ldots, m$ to be from its correct margin boundary.

$$\begin{aligned} &\mathbf{w}, b = \underset{\mathbf{w},b,\boldsymbol{\zeta}}{\arg\min}\ \frac{1}{2}||\mathbf{w}||^2 + \sum_{i=1}^{m} \zeta_i \\ &\text{subject to: } y_i(\mathbf{w}\phi(\mathbf{T}_i + b)) \geq 1 - \zeta_i, \\ &\qquad\qquad \zeta_i \geq 0,\ i = 1, \ldots, m \end{aligned} \tag{4.5}$$

The optimization problem found in Equation 4.5 can be transformed into the problem found in Equation 4.6 with the use of Lagrange multipliers.

$$\begin{aligned} &\boldsymbol{\alpha} = \underset{\boldsymbol{\alpha}}{\arg\min}\ \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}\,\boldsymbol{\alpha} \\ &\text{subject to: } \mathbf{y}\boldsymbol{\alpha} = 0, \\ &\qquad\qquad 0 \leq \alpha_i \leq 1,\ i = 1, \ldots, m \end{aligned} \tag{4.6}$$

In Equation 4.6 $\mathbf{e} \in \mathbb{R}^m$ is a vector of all ones, and m is equal to the number of samples in the dataset $\mathcal{X}$ for which the model is optimised. Secondly, $\boldsymbol{\alpha} \in \mathbb{R}^m$ is the vector of the Lagrange multipliers α_i for $i = 1, \ldots, m$. Lastly, Q is a $m \times m$ positive semi-definite matrix, where the entries are given by $Q_{ij} \equiv y_i y_j K(\mathbf{T}_i, \mathbf{T}_j)$ for all $i, j \in \{1, 2, \ldots, m\}$. The function K used when computing the entries of Q is a kernel function, that is $K(\mathbf{T}_i, \mathbf{T}_j) = \phi(\mathbf{T}_i)\phi(\mathbf{T}_j)$. The kernel trick is used when computing the Kernel function. This allows for the dot product to be computed between two vectors that are mapped to a higher dimensional space, without explicitly computing the mapping to that space.

When the optimisation problem, found in Equation 4.6, has been solved we can predict the class values of new samples. This is done by applying the sign function to the updated decision boundary function db' which is given by the following equation.

$$\begin{aligned} &\hat{y}_{new} = \begin{cases} 0, & sign(db'(\mathbf{T}_{new}) = -1 \\ 1, & \text{otherwise} \end{cases} \\ &db'(\mathbf{T}_{new}) = \sum y_i \alpha_i K(\mathbf{T}_i, \mathbf{T}_{new}) + b,\ \forall \mathbf{T}_i \in \mathcal{S} \end{aligned} \tag{4.7}$$

In the updated decision boundary function db' in Equation 4.7 we sum over support vectors and their respective target values and Lagrange multipliers. Support vectors are the samples, from the dataset $\mathcal{X}$ used when solving Equation 4.6, that lies within the margin. The collection of support vectors is denoted $\mathcal{S}$. Only the support vectors are used in db' as the Lagrange multiplier, α, is zero for all other samples.

4.4 Recurrent Neural Network

We will, in this section, present the three types of RNN models used in our experimentation: Simple RNN, LSTM, and GRU. We focus on the internal structure and the computations performed on the samples given as input. These samples are part of a dataset $\mathcal{X} = \{\mathbf{T}_i | \forall i \in \{1, 2, \ldots, m\}\}$ of m samples. Similarly to Section 4.3 all time-series within the dataset are of the same length n. However, when using the models presented in this section the dataset is no longer restricted to univariate time-series. Instead, the real-valued vectors of all time-series within the dataset should have the same dimensionality d (Definition 4.1.1). Finally, the loss function and optimizer we use to train these three models in our experimentation are presented in Section 4.4.4.

4.4.1 Simple Recurrent Neural Network

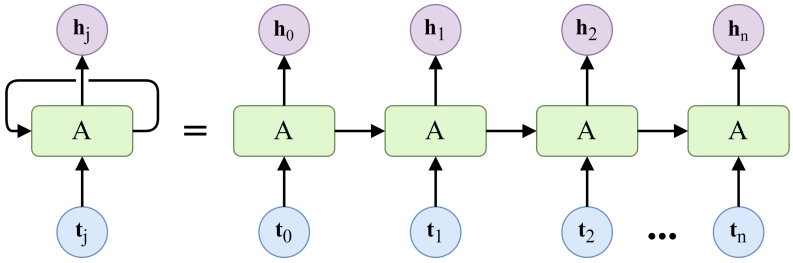


Figure 4.2. Visualisation of the chain-like structure of the simple RNN [26].

The Simple RNN refers to the model introduced by Elman in 1990 [11]. This model has a chain-like structure of repeating modules called hidden cells (Figure 4.2). In this structure each time step $\mathbf{t}_j$ for $j = 1, \ldots n$ of a time-series $\mathbf{T}$ are fed sequentially to the hidden cells together with the output of the previous hidden cell. Thus the output of a hidden cell $\mathbf{h}_j \in \mathbb{R}^h$ at time step j is computed using Equation 4.8. Here h is the number of hidden units within the cell. To compute the output of a hidden cell, two separate weight matrices are used namely W and U. The weight matrix $W \in \mathbb{R}^{h \times d}$ is applied to the current time-step $\mathbf{t}_j$. Thus d is equal to the dimensionality of the time-steps within $\mathbf{T}$ (Definition 4.1.1). The matrix $U \in \mathbb{R}^{h \times h}$ is on the other hand applied to the output of the previous hidden cell. Finally, $\mathbf{b} \in \mathbb{R}^h$ is the bias vector and $tanh$ is the hyperbolic tangent function. [27]

$$\mathbf{h}_j = tanh(W\mathbf{t}_j + U\mathbf{h}_{j-1} + \mathbf{b}) \text{ for } j = 1, \ldots, n \text{ where } \mathbf{h}_0 = \mathbf{0} \tag{4.8}$$

The weights of the model are updated at each time step by doing backpropagation through time (BPTT) on the output of the current time step [28]. With BPTT the RNN model is unfolded into a feed-forward NN and backpropagation is applied to update the weights.

When using backpropagation the error between the predicted output and the target value is first calculated and propagated backwards through the network. Hereafter, the gradients obtained from the backward pass through the network are used to update the weights.

The Simple RNN can quickly be affected by vanishing or exploding gradients. This is due to the multiplication of gradients when applying backpropagation backwards through time. Vanishing gradients prevents the model from learning long-term dependencies while exploding gradients can cause the model to become unstable.[27]

4.4.2 Long Short Term Memory

This section is based on [3, 29]. The LSTM model was first presented in Hochreiter and Schmidhuber [12]. By introducing 'gates' within the hidden cells of the simple RNN structure, the LSTM model could mitigate the vanishing/exploding gradient problem.

Similar to the simple RNN model the LSTM model also has a chain-like structure. However, the hidden cells of this model are more complex due to the introduction of gates. In Figure 4.3 we see a representation of the inner workings of these hidden cells.

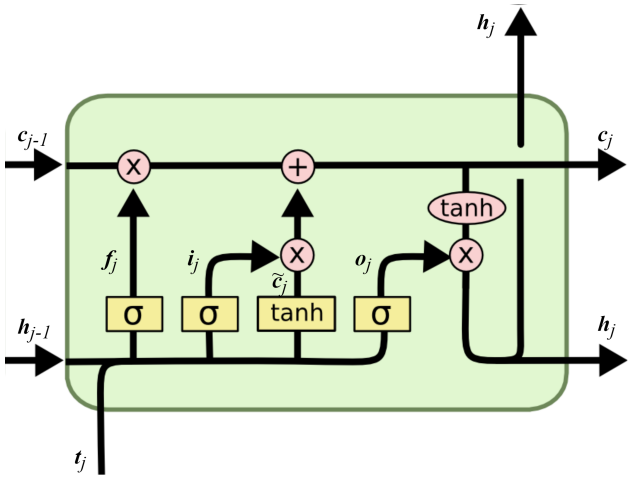


Figure 4.3. The inner workings of a LSTM hidden cell [3].

To understand the computations within a hidden cell we extend the notation used in Section 4.4.1. As can be seen in Figure 4.3 a hidden cell at time step j for $j = 1, \dots n$ now has three inputs rather than the usual two. These are:

- The output vector of the previous hidden cell: $\mathbf{h}_{j-1} \in \mathbb{R}^h$.
- The current time step vector $\mathbf{t}_j \in \mathbb{R}^d$ of the input time-series $\mathbf{T}$ of length n.
- The cell state of the previous hidden cell $\mathbf{c}_{j-1} \in \mathbb{R}^h$

The hidden cells of the LSTM model consist of four layers namely: the forget gate f, input gate i, output gate o, and the memory cell layer c. Consequently, the hidden cells make use of eight weight matrices and four bias vectors. The weight matrices applied to the input vector and the output of the previous hidden cell are respectively denoted as $W_q \in \mathbb{R}^{h \times d}$ and $U_q \in \mathbb{R}^{h \times h}$ for $q \in \{f, i, o, c\}$. The four bias vectors are denoted as $\mathbf{b}_q \in \mathbb{R}^h$ for $q \in \{f, i, o, c\}$. Within this notation $q \in \{f, i, o, c\}$ indicates which of the four layers the matrices and bias vectors belong to. Additionally, the LSTM model utilises the two activation functions: sigmoid σ and the tangent hyperbolic function *tanh*. Finally, it applies to all equations in this section that $\mathbf{h}_0 = \mathbf{0}$.

The computations within the hidden cell can intuitively be divided into two parts. In the first part, the cell state is updated and in the second part, the output of the hidden cell is computed.

Updating the cell state is done by first evaluating what information to be disregarded from the cell state of the previous hidden cell $\mathbf{c}_{j-1}$ (Equation 4.9). Hence the output vector $\mathbf{f}_j$ from Equation 4.9 represents the importance of the elements within $\mathbf{c}_{j-1}$. The next step is to determine the new information to be stored in the cell state. In Equation 4.10 the vector $\mathbf{i}_j$ is computed. This vector indicates which values of $\mathbf{c}_{j-1}$ that should be updated with new information. Furthermore, the vector $\tilde{\mathbf{c}}_j$, computed in Equation 4.11, contains the new candidate values to update the cell state with. Finally, the cell state $\mathbf{c}_j$ is computed using Equation 4.12. Note that $\odot$ is the Hadamard product, also known as the elementwise product.

$$\mathbf{f}_j = \sigma(W_f \mathbf{t}_j + U_f \mathbf{h}_{j-1} + \mathbf{b}_f) \tag{4.9}$$
$$\mathbf{i}_j = \sigma(W_i \mathbf{t}_j + U_i \mathbf{h}_{j-1} + \mathbf{b}_i) \tag{4.10}$$
$$\tilde{\mathbf{c}}_j = tanh(W_c \mathbf{t}_j + U_c \mathbf{h}_{j-1} + \mathbf{b}_c) \tag{4.11}$$
$$\mathbf{c}_j = \mathbf{f}_j \odot \mathbf{c}_{j-1} + \mathbf{i}_j \odot \tilde{\mathbf{c}}_j \tag{4.12}$$

Computing the output of the hidden cell $\mathbf{h}_j$ is a straightforward process when first the cell state has been updated. First, the output gate is computed using Equation 4.13. The results from this act as a form of short-term memory. Combining this with the current state of the acting long-term memory $\mathbf{c}_j$ we see the reason for the name of this model. The output of the current hidden cell at time-step j for $j = 1, \ldots, n$ is as such computed using Equation 4.14.

$$\mathbf{o}_j = \sigma(W_o \mathbf{t}_j + U_o \mathbf{h}_{j-1} + \mathbf{b}_o) \tag{4.13}$$
$$\mathbf{h}_j = \mathbf{o}_j \odot tanh(\mathbf{c}_j) \tag{4.14}$$

4.4.3 Gated Recurrent Unit

The GRU model was first presented in Cho et al. [30]. Similar to the LSTM model this model also introduces 'gates' to the hidden cells of the simple RNN. However, different from the LSTM model the GRU model does not have a separate cell state [1]. Similar to the simple RNN and LSTM models the GRU model also has a chain-like structure. In Figure 4.4 we see a representation of the inner workings of the hidden cells of this model.

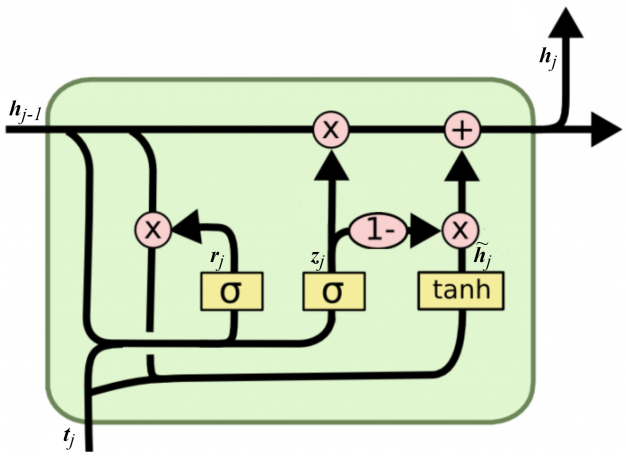


Figure 4.4. The inner workings of a GRU hidden cell [31].

[1] $\mathbf{c}_j$ for $j = 1, \ldots, n$ from the LSTM model (Section 4.4.2)

The notation used when presenting the maths used within GRU is mostly similar to that found in Section 4.4.2. Although, due to the different layers within this model, there are minor changes in the notation.

The GRU model has three layers namely: the reset gate r, the update gate z, and the candidate layer $\tilde{h}$. Thus, the weight matrices applied to the input vector and the output of the previous layer are respectively denoted as $W_s \in \mathbb{R}^{h \times d}$ and $U_s \in \mathbb{R}^{h \times h}$ for $s \in \{r, z, \tilde{h}\}$. Likewise, are the bias vectors denoted as $\mathbf{b}_s \in \mathbb{R}^h$ for $s \in \{r, z, \tilde{h}\}$. Similar to q in Section 4.4.2, s indicates for which of the three layers the weight matrices and bias vectors belong. Finally, recall that $\mathbf{h}_0 = \mathbf{0}$.

The reset gate of a hidden cell at time step j is computed using Equation 4.15. Given this, the candidate layer can be computed (Equation 4.16). From Equation 4.16 we can see that if the values of the reset gate are close to 0 the output of the previous hidden cell is ignored. If this is the case the information is then 'reset' with the current input $\mathbf{t}_j$ instead.

$$\mathbf{r}_j = \sigma(W_r \mathbf{t}_j + U_r \mathbf{h}_{j-1} + \mathbf{b}_r) \tag{4.15}$$

$$\tilde{\mathbf{h}}_j = tanh(W_{\tilde{h}} \mathbf{t}_j + U_{\tilde{h}}(r \odot \mathbf{h}_{j-1}) + \mathbf{b}_{\tilde{h}}) \tag{4.16}$$

The final gate before computing the output of the hidden cell is the update gate. This gate is computed by applying Equation 4.17. Given the update vector $\mathbf{z}_j$ the output of the current hidden cell is found by using Equation 4.18. The update gate thus controls how much information from the output of the previous hidden cell is to be used in the current hidden cell.

$$\mathbf{z}_j = \sigma(W_z \mathbf{t}_j + U_z \mathbf{h}_{j-1} + \mathbf{b}_z) \tag{4.17}$$

$$\mathbf{h}_j = \mathbf{z}_j \odot \mathbf{h}_{j-1} + (1 - \mathbf{z}_j) \odot \tilde{\mathbf{h}}_j \tag{4.18}$$

4.4.4 Adam Optimizer

Optimizer algorithms are used to update the weights and biases of a model over a number of epochs to minimise the loss function. Loss functions are used to measure the difference between the predicted values $\hat{\mathbf{y}} \in \mathbb{R}^m$ and the target values $\mathbf{y} \in \mathbb{R}^m$ of a model given a dataset $\mathcal{X}$ of m samples. The loss function commonly used for binary classification is the binary cross-entropy, also known as log-loss [32]. Binary cross-entropy is given by Equation 4.19. Instead of using the predicted target values, Equation 4.19 use the predicted probability of a sample belonging to class 1. That is $\hat{p}_i = P(y_i = 1 | \mathbf{T}_i)$ for all $\mathbf{T}_i \in \mathcal{X}$.[3, 33]

$$log\text{-}loss = -\frac{1}{m} \sum_{i=1}^{m} y_i \cdot log(p_i) + (1 - y_i) \cdot log(1 - p_i) \tag{4.19}$$

Most optimization algorithms utilise gradient descent. The intuition of which is to calculate the gradients in the model by computing the derivative of the loss function. These gradients are then used to determine the direction in which the weights and biases are updated.

The Adaptive Moment Estimation (Adam) algorithm[34] is an optimization algorithm that makes use of an adaptive learning rate together with momentum. These elements are introduced to help the model converge on a local minima faster. The learning rate at iteration t of the optimisation is determined for each model parameter $\mathbf{w}$ by first calculating the exponential moving average $\mathbf{m}$ and the squared gradient $\mathbf{v}$ as seen in Equation 4.20.

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \qquad \mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \tag{4.20}$$

Here, $\mathbf{g}_t$ is the gradient of $\mathbf{w}$ at time t, and $\mathbf{g}^2 = \mathbf{g} \odot \mathbf{g}$. β_1 and β_2 are the default parameters of 0.9 and 0.999 respectively. After calculating $\mathbf{m_t}$ and $\mathbf{v_t}$, they are both corrected for bias to prevent them from converging towards zero. The bias correction of $\mathbf{m_t}$ and $\mathbf{v_t}$ can be seen in Equation 4.21.

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t} \qquad \hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t} \tag{4.21}$$

The bias-corrected moving averages, $\hat{\mathbf{m}}_t$ and $\hat{\mathbf{v}}_t$, are finally used to update the model parameters $\mathbf{w}$ at time t as shown in Equation 4.22.

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \varepsilon} \tag{4.22}$$

Here, η is the learning rate and ε is a near zero constant to avoid zero division. [3, 34]

4.5 Time-series Classification

Time-series classification is another group of machine learning models for sequential data. These models are as their names imply specialised in classifying time-series. The models we focus on are ROCKET[17] and HC2[18]. In this section, we present the intuition of these two models. Note that for these models, the dataset $\mathcal{X}$ consists of m univariate time-series all of length n.

4.5.1 Random Convolutional Kernel Transform

ROCKET is a time-series classification model that has been shown to be both fast and accurate compared to other state-of-the-art models. ROCKET works by initialising 10 000 random convolutional kernels that are used to transform all time-series from the dataset $\mathcal{X}$. These transformed features are then used to train a linear classifier We will here give a notion of the kernel creations and how they are used to transform the data before using it as input to a logistic regression model.[17]

Applying a convolutional kernel to some data produces a feature map that highlights certain relevant features of the input data. The convolutional kernels generated and used in ROCKET consist of five different parameters that are initialised with random values as follows:

- The *length* of the kernel l_{kernel} is with equal probability selected from the set $\{7, 9, 11\}$.
- The *weights* of the kernel ω are first sampled from the normal distribution $\mathcal{N}(0, 1)$ and thereafter mean centred.
- The kernel *bias* b is sampled from the uniform distribution $\mathcal{U}(-1, 1)$.
- The *dilation* of the kernel is $d = \lfloor 2^x \rfloor$ where x is sampled from the uniform distribution $\mathcal{U}(0, A)$ and $A = log_2 \frac{l_{input}-1}{l_{kernel}-1}$. Here l_{input} is the length of the time-series $\mathbf{T}$ given as input.
- The *padding* is a boolean value of which the value is chosen randomly.

Note that two otherwise similar kernels except for their dilation to match the same or similar patterns at different frequencies. Furthermore, if padding is chosen to be used for a kernel the time-series $\mathbf{T}$ given as input is appended at the start and end with a zero-padding. This is done such that the 'middle' element of the kernel is centred on every point in $\mathbf{T}$.

When the convolutional kernels have been randomly generated they are applied to each time-series of length n within the dataset $\mathcal{X}$ which results in a number of feature maps for each time-series. Computing the feature map given a time-series $\mathbf{T}_i$ and kernel ω is done using Equation 4.23.

$$\mathbf{f} = \left\langle \sum_{j=0}^{l_{kernel}-1} t_{i+j\times d} \times \omega_j | i = 1, \dots n \right\rangle \tag{4.23}$$

For each feature map $\mathbf{f}$ for a time-series $\mathbf{T}_i$, ROCKET computes two features. First is the maximum value of the computed feature map max. Second, is the proportion of positive values ppv which is the number of positive entries in $\mathbf{f}$. The final transform of $\mathbf{T}_i$ is then $\mathbf{k}_i = \langle (max_c, ppv_c) | c = 1, \dots 10000 \rangle$ where c indicates the kernel used.

As only positive values in the computed features maps are ultimately used, the aforementioned bias allows for two otherwise similar kernels to highlight different aspects. This effect happens as the bias shifts the values above or below zero with a fixed amount.

4.5.2 HIVE-COTE 2.0

HC2 is a state-of-the-art ensemble model for time-series classification. The ensemble is comprised of four main components that are trained independently. These four components are Shapelet Transform Classifier (STC), ROCKET, Temporal Dictionary Ensemble (TDE) and Diverse Representation Canonical Interval Forest (DrCIF). This section describes the intuition behind these different components as HC2 is quickly discarded as a primary focus of the experimentation (Section 6.3.3). [18]

STC[35]: Shapelets, within time-series classification, refer to sub-sequences of a time-series that is used to discriminate classes. STC is a model that searches for high-quality shapelets across multiple time-series independent of position and alignment. Here it starts by sampling a set number of shapelets for each time-series. The quality of these shapelets is then calculated using information gain to determine how well a shapelet can discriminate the classes of the dataset. Top-k shapelets are then used to train a classifier. [18, 35, 36]

ROCKET[17]: HC2 implements the ROCKET model described in Section 4.5.1. HC2 implements an ensemble of multiple ROCKET classifiers with fewer kernels to get better class probabilities that better fit HC2. [17, 18]

TDE[37]: TDE is an ensemble approach of dictionary classifiers that uses the bag-of-words concept. The ensemble of TDE consists of dictionaries that are generated with different parameters such as word length and window size. Words are determined by a sliding window of a given length over a time-series. Occurrences of each word per time-series are counted and the resulting histogram is used to train a classifier. [18, 37]

DrCIF[38]: DrCIF is an interval-based classifier ensemble. Interval-based classifiers work by extracting subsequences from time-series to discriminate classes, similar to shapelets from STC. The main difference between these two approaches is that interval-based classifiers work on phase-dependent intervals, meaning that the position and alignment of the subsequences are taken into consideration. The quality of a given interval in terms of its ability to discriminate classes is determined by information gain. The DrCIF ensemble consists of multiple interval-based classifiers that use different intervals. [18, 38]

HC2 uses Bootstrap Aggregation (Bagging) to combine the probability predictions for all classifiers previously described. Bagging includes independently training the classifiers in parallel and averaging the resulting probability predictions. [18]

4.6 Evaluation Metrics

Evaluation metrics are used to measure the performance of, in this case, a machine learning model. There exist a variety of different evaluation metrics. However, in this study, we use accuracy and F1-score. [3, 39]

Before presenting the equations of the accuracy and F1 metrics for the binary classification problem we describe four key terms. These are: true positives TP, true negatives TN, false positives FP, and false negatives FN. TP and TN are the number of samples that are correctly predicted to have the positive and negative label respectively. On the other hand, FP and FN are the number of samples wrongly predicted to have the positive and negative label respectively.

Accuracy is calculated as the ratio between the number of correct predictions and the total number of predictions, see Equation 4.24.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.24}$$

The F1-score is generally used when the dataset is imbalanced as it takes the label distribution into account. It is calculated by taking the harmonic mean of the two metrics precision and recall, see Equation 4.25. The equations for precision and recall can be seen in Equation 4.26. Here we see that the precision is the proportion of the positive predictions made that was correct. The recall is then the proportion of positive samples that were predicted correctly.

$$F1\text{-}score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{4.25}$$

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN} \tag{4.26}$$

Modelling 5

Given the dataset introduction (Chapter 3) and background theory (Chapter 4), we present relevant implementation details of the experimental setup in this chapter. Our experimental setup is implemented in Python and any following implementation details are consequently Python related. The purpose of this chapter is to ensure transparency and reproducibility of the experiments performed in this study.

We first introduce how the data is preprocessed and initialised to match the expected input of the machine learning models. Next, the necessary implementation details of the different models are presented.

5.1 Preprocessing

The raw log data from the datasets are transformed to match the expected input shape of the models tested in the experiments. This is done by extracting the high word values from log entries with relevant CAN-IDs and then constructing a time-series given these high word values. The expected input shape depends on the model and in our case results in two scenarios. The first scenario is a 2-dimensional Numpy array with the shape `(Number of samples, Number of time-steps)`. In the second scenario, a third dimension is added when using high word values from multiple CAN IDs. Thus the time-series created is multivariate. This 3-dimensional Numpy array has the shape `(Number of samples, Number of time-steps, Number of features)`. The 3-dimensional array is only used as input to the RNN models: SimpleRNN, LSTM, and GRU. The 2-dimensional array is then used as input to the other models LR, SVM, ROCKET, and HC2.

5.1.1 Matrix Profiles

The creation of matrix profiles is an optional part of the preprocessing. In order to create these we use the '`matrixprofile.compute`' function from the MatrixProfile library[10]. Given a list of window sizes, the respective matrix profiles are computed for each time-series in the dataset. If this list of window sizes is larger than one, the newly created matrix profiles are zero-padded at the end so that they all match the length of the largest profile. Hereafter, the multiple matrix profiles for one time-series are combined to create a multivariate time-series. As an example let the list of window-sizes be $[5, 10, 25]$ then given a univariate time-series $\mathbf{T}$ the matrix profiles $\mathbf{M}_5$, $\mathbf{A}_{10}$, and $\mathbf{R}_{25}$ are individually computed. Hereafter zero padding is used such that all three profiles have the same length n[1]. Finally the three profiles are combined such that $\mathbf{P}_{[5,10,25]} = \mathbf{p}_1, \ldots, \mathbf{p}_n$ where $\mathbf{p}_i = \langle m_i, a_i, r_i \rangle$ for

[1] n is the length of the largest profile before applying zero-padding

$1 \leq i \leq n$. Here m_i, a_i, and r_i are the i'th entry in the zero padded matrix profiles $\mathbf{M}_5$, $\mathbf{A}_{10}$, and $\mathbf{R}_{25}$ respectively.

The output of this step has one of two shapes. If the size of the list of window sizes is one, a univariate time-series is created for each time-series in the input. The final output is thus a 2-dimensional Numpy array with the shape `(Number of samples, Number of time-steps)`. Where the number of time-steps is equal to the length of the matrix profile. On the other hand, if the list of window sizes is larger than one, a multi-variate time-series is created instead for each time-series in the dataset. The output in this case is the 3-dimensional Numpy array with the shape `(Number of samples, Number of time-steps, Number of window-sizes)`.

5.2 Baseline Models

The two baseline models used in this study are LR and SVM. These models are implemented using the sci-kit learn library[24]. Experiments using LR uses sci-kits '`sklearn.linear_model.LogisticRegression`' classifier using the L-BFGS solver together with the L2-regularization term. Experiments using SVM use sci-kits '`sklearn.svm.SVC`' with the 'rbf' kernel. The parameters for both LR and SVM are the default parameters for the respective sci-kit implementations. As presented in Section 5.1, these models use the 2-dimensional array representation as input.

5.3 RNN Models

We use the Keras library[40] to build the RNN models tested in our experiments. This includes defining the layers of the models together with specifying their respective parameters. We implemented the function `build_rnn()` (Listing 5.1) to streamline the modelling process and quickly build a variety of models with different architectures. It builds a model with a specific architecture given a model type (`model_name`) and the number of consecutive RNN layers (`num_layers`)(lines 4-13). The first layer of this new model will consist of a user-defined number of units (`init_units`). Thereafter this number is halved for each consecutive layer (line 13). This approach to modelling the architectures was chosen to conform to standard practices within machine learning [41].

The three RNN layer types supported in `build_rnn()` are SimpleRNN, LSTM and GRU. They are implemented by setting the `layer` variable in `build_rnn()` to one of the respective layer classes from Keras (line 3), listed in Table 5.1.

Layer Type	**Layer Class**
SimpleRNN	`tf.keras.layers.SimpleRNN()`
LSTM	`tf.keras.layers.LSTM()`
GRU	`tf.keras.layers.GRU()`

Table 5.1. Keras layer classes used for the different RNN layer types.

Models that have more than one RNN layer are set to pass all hidden states of the current RNN layer to the next using the '`return_sequence`' flag (line 7 and 11). The models will

```
1  def build_rnn(data_shape, model_name, init_units, num_layers, min_units):
2      model = Sequential()
3      layer = eval(model_name)
4      for i in range(num_layers):
5          if i == 0: # First layer need to specify input shape.
6              model.add(layer(units,
7                              return_sequences=(i < num_layers - 1),
8                              input_shape=(data_shape[1], data_shape[2])))
9          else:
10             model.add(layer(units,
11                             return_sequences=(i < num_layers - 1),))
12         if units > min_units:
13             units //= 2
14     model.add(Dense(1, activation='sigmoid'))
15     model.compile(loss='binary_crossentropy',
16                   optimizer='adam',
17                   metrics=['accuracy', get_f1])
18     return model
```

Listing 5.1. `build_rnn()` implemenation.

always end with a dense layer of size 1 (line 14). This layer uses the sigmoid activation function for the model to output a probability used for the binary classification. All RNN models build with `build_rnn()` use the Adam optimizer together with the binary cross-entropy loss function (line 15).

5.4 Time-series Classification

The two time-series classification methods tested in this project are ROCKET and HC2. These are implemented using the Sktime library[42]. To build the ROCKET models we use the '`sktime.transformations.panel.rocket.Rocket`' class. Here, 10 000 kernels are randomly initialised and the normalisation flag is set to true. These parameters are chosen as they are the default values and have been shown to generally perform the best[17]. The HC2 models are implemented using the '`sktime.classification.hybrid.HIVECOTEV2`' class with default parameters.

Experimentation 6

This chapter goes through our experimentation, including the experimental setting, range of experiments and result analysis. The implementation of the experimental setup is described in Chapter 5.

6.1 Experimental Setting

The experimental methodology for this study is to control for confounding variables and blocking factors as much as possible while manipulating independent variables to examine their effect on the results. This is in part achieved by introducing a range of control constants across the different experiments.

The control constants include using the same input data across experiment groups. Here, the *original dataset* refers to the "LANSCE 18 Cruise" dataset (Section 3.1.1) with only unique samples and excluding the three outliers mentioned in Section 3.2.3. The reason we only looked at unique samples for the original dataset is based on the findings of Mikkelsen and Bonde [3]. This study concludes that using the full dataset adds unwanted bias to the models compared to only using unique samples. The second dataset used is the *fail-injected dataset* which refers to a dataset created by combining the nominal- and fail-injected extensions (sections 3.1.2 and 3.1.3). Here 20 000 samples have been randomly sampled from each dataset resulting in a dataset with a total of 40 000 samples. This number of simulated samples was chosen to balance the amount of information and time complexity. The exact number of samples for the two datasets can be found in Table 6.1. Throughout the experimentation and regardless of the dataset used, 20% of the data is set aside for testing, leaving 80% of the data for training the models.

Dataset Name	**Number of negative samples**	**Number of positive samples**	**Total number of samples**
Original dataset(†)	52	56	108
Fail-injected dataset(‡)	20 000	20 000	40 000

Table 6.1. Dataset specifications.

We consider two input features from the datasets, namely the vehicle speed and torque. All models, unless otherwise stated, are trained on both of these input features individually as univariate time-series. Some models are also trained using both input features at the same time as multivariate time-series. These three variations of the data representation are specified in Table 6.2.

Each experiment is replicated three times independently, where the only difference is the seed. The three arbitrary seeds 42, 404, and 192 are used in all experiments to control for

Input Feature	Description
Vehicle Speed	Univariate time-series constructed from the logged high word values for entries with the CAN ID "PLANT_INFO_VehicleSpeed".
Torque	Univariate time-series constructed from the logged high word values for entries with the CAN ID "CTRL_INFO_TorqueCommand".
Vehicle Speed & Toruqe	Multi-variate time-series constructed from the logged high word values for entries with the CAN ID "PLANT_INFO_VehicleSpeed" and "CTRL_INFO_TorqueCommand".

Table 6.2. Input feature specifications.

confounding variables and ensure reproducibility. These seeds are used in different parts of the code such as data split, shuffling of data, and weight initialisation. The arithmetic mean and standard deviation of the results of the three runs are computed and presented in Section 6.3. The results from the individual runs of each experiment can be found in Appendix D.

The remaining control constants specific to the different groups of experiments are covered in Section 6.2.

6.2 Experiment Groups

The experiments conducted in this study can be divided into four main categories. These are: baseline, RNN, time-series classification, and matrix profile. This section will explain the experimental setup as well as the range of experiments within each category.

6.2.1 Baseline Models

We have chosen two traditional machine-learning models as our baselines, these are LR and SVM. These two models are chosen to give an insight into the performance of relatively simple models on this binary classification problem. They can further be used to argue for the need for more complex/specialised solutions.

These baseline models are run with different combinations of the datasets and single input features. Thus, we individually train the models on the original and simulated dataset with the two univariate input feature variations.

6.2.2 RNN

The RNN experiments include the testing of different RNN models, input data and architectures. The RNN models tested are SimpleRNN, LSTM, and GRU. Each of these models are tested on three specified architectures modelled with the `build_rnn()` function described in Section 5.3. The specification of the three architectures can be seen in Table 6.3. These architectures are chosen to determine the effect that the size of the RNN model has going from a small model (A1) to a relatively large model (A3). The

combinations of model types and architectures are also all tested on the three different variations of input features.

Name	init_units	num_layers	Description
Architecture 1 (A1)	32	1	A small model with one RNN layer of 32 units.
Architecture 2 (A2)	64	2	A medium sized model with two consecutive RNN layers of 64 and 32 units respectively.
Architecture 3 (A3)	128	3	A large model with three consecutive RNN layers of 128, 64 and 32 units respectively.

Table 6.3. Architecture specifications.

All RNN experiments are trained for 150 epochs as initial experiments, given their loss curves, showed this to be enough to converge. Other control constants include training on batch sizes of 32 and an initial learning rate of 0.001. These control constants were chosen as they are the default parameters for the Adam optimizer and were not the focus of our RNN experimentation [34]. All experiments were run on the original dataset. LSTM and GRU are further run on the simulated datasets as well. The SimpleRNN is not included in these experiments. This is due to the fact that the Keras implementation is not GPU accelerated resulting in too long training times.

6.2.3 Time-series Classification

The time-series classification experiments include testing of the two presented methods, ROCKET and HC2, on the different combinations of datasets and univariate input features. However, HC2 was not tested on the simulated dataset due to too long training times. The goal of these experiments is to examine whether these more specialised models can outperform the more generally applicable baseline and RNN models tested.

6.2.4 Matrix Profile

Experiments within the matrix profile group refer to models that are trained using matrix profiles generated using the window sizes $L \in \{5, 15, 25, 100\}$ as input. These inputs are generated using the approach described in Section 5.1.1. We initially experiment with combinations of the two datasets and different matrix profiles as input to the baseline models LR and SVM. This is done to get an initial insight into the difference in performance compared to training the models on the original logged data as well as determining the best-performing window sizes.

The matrix profiles with the window sizes that performed the best on the baseline models are then applied to a select number of models that previously have shown potential. For the RNN we also experimented with using the multivariate matrix profiles $\mathbf{P}_{[15,25]}$. Thus combining the matrix profiles $\mathbf{P}_{15}$ and $\mathbf{P}_{25}$ as described in Section 5.1.1. This is again to see whether transforming the data to a matrix profile will result in better overall performance, in terms of accuracy and F1-score, compared to using the original data representation.

6.3 Results

This section presents relevant aggregated results and observations within each experiment group. The non-aggregated result can be found in Appendix D.

6.3.1 Baseline

The aggregated results of the baseline experiments can be seen in Table 6.4. These results are used as a comparison point to the other experiments conducted. The best-performing baseline experiment is the SVM model on the simulated dataset with torque as the input feature. This experiment yielded an accuracy and F1-score of 0.53 and 0.54 respectively.

Two main observations were made given these baseline experiments. Firstly, the SVM model on the original dataset tends to consistently predict either 0 or 1 resulting in an accuracy of ~ 0.50 and an F1-score of ~ 0.33. Secondly, applying the LR and SVM models on the simulated dataset consistently results in an accuracy and F1-score of ~ 0.50 which is similar to random guessing. These two observations are further supported by examining their respective confusion matrices found in Appendix C. These results point to the fact that this binary classification problem requires more complex/specialised models in order to make useful predictions.

	Vehicle Speed		**Torque**	
	Accuracy	**F1-score**	**Accuracy**	**F1-score**
LR†	0.43 (0.09)	0.43 (0.09)	0.38 (0.02)	0.36 (0.03)
SVM†	0.52 (0.0)	0.34 (0.0)	0.51 (0.02)	0.35 (0.02)
LR‡	0.52 (0.01)	0.51 (0.0)	0.52 (0.0)	0.52 (0.0)
SVM‡	0.53 (0.01)	0.5 (0.05)	0.54 (0.01)	0.52 (0.01)

Table 6.4. Mean accuracy and F1-score with standard deviation noted in parenthesis for the baseline experiments on the original dataset (†) and simulated dataset (‡) independently trained using the features vehicle speed and torque.

6.3.2 RNN

The aggregated accuracies for the different RNN experiments can be seen in Table 6.5. We have, in this group of experiments, chosen to only show the accuracy of the experiments to ease the readability of the results. This is further justified as the datasets tested are balanced and we generally see the same tendencies across the accuracies and F1-scores for these experiments. The best performing RNN experiment is the LSTM model with the small architecture (A1) on the original dataset with the multivariate input feature representation. This experiment achieved an accuracy of 0.58.

Overall, we did not see the expected increase in accuracy and F1-score from these more complex RNN models compared to the baseline experiments. We can instead observe some similar tendencies. One of these observations is that the SimpleRNN on the original dataset

always predicts either 0 or 1 similar to the SVM model. We can furthermore see a general tendency of the results to have an accuracy close to 0.50, especially for the experiments trained on the simulated dataset. This is further evidence that the models do not probably learn any useful features seeing how similar these results are to random guessing. Note that the range of tested architectures is relatively small and more specialised architectures could potentially improve performance.

	Vehicle Speed			**Torque**			**Vehicle Speed & Torque**		
	A1	**A2**	**A3**	**A1**	**A2**	**A3**	**A1**	**A2**	**A3**
SimpleRNN†	0.52 (0.0)	0.52 (0.0)	0.52 (0.0)	0.54 (0.05)	0.55 (0.03)	0.54 (0.03)	0.52 (0.05)	0.51 (0.07)	0.48 (0.09)
LSTM†	0.48 (0.05)	0.42 (0.07)	0.41 (0.07)	0.51 (0.07)	0.51 (0.13)	0.52 (0.12)	0.58 (0.09)	0.48 (0.0)	0.4 (0.02)
GRU†	0.47 (0.11)	0.42 (0.07)	0.42 (0.05)	0.57 (0.0)	0.55 (0.03)	0.52 (0.08)	0.55 (0.11)	0.46 (0.03)	0.42 (0.09)
LSTM‡	0.5 (0.01)	0.5 (0.01)	0.5 (0.01)	0.5 (0.01)	0.51 (0.0)	0.5 (0.01)	-	-	-
GRU‡	0.5 (0.01)	0.5 (0.01)	0.5 (0.01)	0.5 (0.01)	0.51 (0.01)	0.5 (0.01)	-	-	-

Table 6.5. Mean of the accuracy with standard deviation noted in parenthesis for the different RNN experiments on the original dataset (†) and simulated dataset (‡) independently trained on the three input feature variations. A1, A2, and A3 refers to the three model architectures specified in Table 6.3.

6.3.3 Time-series Classification

The aggregated results of the time-series classification experiments can be seen in Table 6.6. Here the best-performing experiment is the ROCKET model on the simulated dataset using torque as the input feature. This experiment achieved an accuracy and F1-score of 0.67 and 0.66 respectively which is a noticeable increase from the best-performing baseline and RNN experiments. Other time-series classification experiments showed a similar increase in performance including the HC2 model on the original dataset using torque as the input feature. With this experiment, we achieve an accuracy and F1-score of 0.65 and 0.64 respectively. However, this experiment had a higher inconsistency compared to ROCKET with a standard deviation of 0.13. This could be contributed to the lower number of samples in the original dataset. In conclusion, these experiments show the potential of using more specialised models.

We considered using the simulated dataset as input to HC2 as well but this was discarded due to too long training times.

6.3.4 Matrix-profile

The aggregated accuracies of the initial matrix profile experiments can be seen in Table 6.7. Note that we for this experiment group have chosen to only present accuracies to improve readability. This decision is further justified as we are only working with balanced datasets and generally see the same tendencies across accuracies and the F1-scores. We initially

	Vehicle Speed		Torque	
	Accuracy	**F1-score**	**Accuracy**	**F1-score**
ROCKET†	0.52 (0.05)	0.49 (0.05)	0.35 (0.08)	0.33 (0.06)
HC2†	0.33 (0.03)	0.33 (0.03)	0.65 (0.13)	0.64 (0.13)
ROCKET‡	0.64 (0.01)	0.63 (0.01)	0.67 (0.01)	0.66 (0.0)

Table 6.6. Mean accuracy and F1-score with standard deviation noted in parenthesis for the time-series classification experiments on the original dataset (†) and simulated dataset (‡) independently trained on the two univariate input features vehicle speed and torque.

ran the two baseline models LR and SVM on the different combinations of datasets, input features, and univariate matrix profiles with different window sizes.

One of the key observations made here is that the initial experiments trained with the simulated dataset, using torque as the input feature saw a noticeable increase in accuracies compared to their respective baseline results. Here the best-performing experiment is the SVM model where the window size of the matrix profiles is set to 15, which yields an accuracy of 0.77. This is a 42.86% percentage increase compared to the baseline experiment of the same experimental setup.

The other initial experiments, namely where the original dataset or the input feature vehicle speed is used, saw similar performance to that of their respective baseline results. For these experiments, we see accuracies ranging from 0.35 to 0.57. The reason why the experiments trained on the original dataset did not see an increase in performance is likely due to a lack of information as a consequence of the small number of samples. The reason that matrix profiles generated using the vehicle speed feature perform worse than those generated from the torque can be attributed to the fact that vehicle speed is affected by inertia making the discords less noticeable. Based on these initial observations, we narrowed the second part of the matrix profile experiments to mainly focus on different models on the simulated dataset and matrix profiles generated given the torque as input feature.

In the second part we experimented with the time-series classification model ROCKET, the results of which can be found in Table 6.7. Here we observed an overall increase in accuracies compared to the initial matrix profile experiments. The best-performing experiment with ROCKET is where $\mathbf{P}_{25}$ for the simulated dataset using torque as input feature. This experiment yielded an accuracy of 0.96 which is a percentage increase of about 43% compared to its respective experiment with the original data representation. This increase in accuracy was somewhat expected as ROCKET is similar to SVM in that they both utilise kernels.

We lastly experimented with using matrix profiles on the previously tested RNN models to see if it would add useful information to these models as well. The aggregated results of the RNN matrix profile experiments can be seen in Table 6.8. Here we chose to only focus on torque as the input feature in combination with the two best-performing window sizes $L = 15, 25$. We also further experimented with combining the generated matrix profiles $\mathbf{P}_{[15,25]}$. The main observation of these experiments is that they show some large variations

	Vehicle Speed				Torque			
	$\mathbf{P}_5$	$\mathbf{P}_{15}$	$\mathbf{P}_{25}$	$\mathbf{P}_{100}$	$\mathbf{P}_5$	$\mathbf{P}_{15}$	$\mathbf{P}_{25}$	$\mathbf{P}_{100}$
LR†	0.39 (0.09)	0.41 (0.07)	0.32 (0.03)	0.49 (0.07)	0.35 (0.05)	0.39 (0.08)	0.43 (0.14)	0.42 (0.02)
SVM†	0.46 (0.03)	0.46 (0.05)	0.46 (0.05)	0.46 (0.05)	0.45 (0.09)	0.56 (0.08)	0.55 (0.05)	0.57 (0.05)
ROCKET†	-	-	-	-	0.64 (0.11)	0.54 (0.09)	0.65 (0.05)	0.37 (0.07)
LR‡	0.5 (0.01)	0.51 (0.01)	0.51 (0.0)	0.51 (0.0)	0.63 (0.01)	0.66 (0.0)	0.66 (0.0)	0.54 (0.0)
SVM‡	0.52 (0.01)	0.52 (0.01)	0.51 (0.01)	0.54 (0.01)	0.75 (0.01)	0.77 (0.0)	0.75 (0.0)	0.65 (0.01)
ROCKET‡	-	-	-	-	0.89 (0.01)	0.88 (0.0)	0.96 (0.01)	0.77 (0.0)

Table 6.7. Mean accuracy with the standard deviation noted parenthesis for the matrix profile (P_L) experiments of different windows sizes L for the matrix profile $\mathbf{P}_L$ on the original dataset (†) and simulated dataset (‡) independently trained using the two univariate input features vehicle speed and torque.

in the results across the three individual runs of each experiment. However, some specific experiments show promising results with accuracies up to 0.99. This result was achieved by the GRU model with architecture 2 (A2) on the simulated dataset with the multivariate matrix profile $\mathbf{P}_{[15,25]}$. This is a percentage increase in the accuracy of about 94% compared to its respective experiment with the original data representation.

The aforementioned inconsistencies appear in two different forms. Firstly, most experiments trained on the simulated dataset have a relatively high standard deviation. Secondly, a handful of experiments trained on the simulated dataset experienced a "Not a Number" error[1] during training, resulting in the training stopping prematurely. The exact reasons for these inconsistencies are unknown but one likely reason would be that some models experience an exploding gradient at some point during training.

	$\mathbf{P}_{15}$			$\mathbf{P}_{25}$			$\mathbf{P}_{[15,25]}$		
	A1	**A2**	**A3**	**A1**	**A2**	**A3**	**A1**	**A2**	**A3**
LSTM†	0.45 (0.07)	0.48 (0.05)	0.48 (0.08)	0.48 (0.05)	0.43 (0.09)	0.49 (0.05)	0.55 (0.14)	0.48 (0.05)	0.48 (0.08)
GRU†	0.52 (0.0)	0.58 (0.1)	0.52 (0.16)	0.51 (0.02)	0.46 (0.07)	0.46 (0.07)	0.48 (0.0)	0.49 (0.07)	0.49 (0.09)
LSTM‡	0.53* (0.05)	0.5 (0.01)	0.5 (0.01)	0.54* (0.11)	0.62 (0.22)	0.5 (0.01)	0.65 (0.18)	0.63 (0.23)	0.5 (0.01)
GRU‡	0.81 (0.17)	0.64* (0.25)	0.64* (0.24)	0.66* (0.15)	0.77 (0.25)	0.66* (0.29)	0.85 (0.16)	0.99 (0.01)	0.66* (0.29)

Table 6.8. Mean accuracy with standard deviation noted in parenthesis for RNN matrix profile experiments on the original dataset (†) and simulated dataset (‡) and different window size L for the matrix profiles $\mathbf{P}_L$. A1, A2, and A3 refer to the three model architectures specified in Table 6.3. Experiments marked with '*' have stopped training prematurely due to a "Not a Number" error.

[1]Experiments affected by the "Not a Number" is marked with '*' in Table 6.8.

7 Discussion

This chapter will shed light on some discussion points and potential future works given our results presented in Section 6.3.

7.1 Result Discussion

The task of classifying abnormal behaviour caused by soft-errors in logs gathered from PID-controlled environments is still new meaning that knowledge of models that would perform well in this domain is limited. Due to this, the main focus of the study has been to test a variety of different machine-learning models that have been applied in similar domains. Parameter tuning of the individual models has thus been down-prioritised. Bad performing models in this study can therefore not be completely discarded based on our results, as parameter tuning could potentially improve the performance. However, the results presented can be used as initial guidance as to what kind of models and input data that have the most potential for further improvements.

Several steps have been taken to ensure the validity of the results presented. One of these steps is that we run each experiment three times with different seeds and calculate the standard deviation between the three runs. Most experiments conducted seem to be relatively consistent as they have standard deviations of less than 0.10. The primary exception is the RNN experiments trained on matrix profiles. Here we both see large variations in the mean and standard deviation of the accuracy of the different models. These inconsistencies across similar experiments can be attributed to the RNN models not being able to train correctly which may be due to an exploding gradient.

Further research within this domain would benefit from additional experimentation. One direction could be to experiment with further tuning of the best performing models by doing a grid search of relevant parameters. The RNN model experiments could also be further extended with more architecture variations and the introduction of mechanisms such as dropout. Finally, the data used as input are not normalised as initial experiments showed it did not result in a noticeable difference in results. However, a more thorough investigation of the effect of normalising the data could be beneficial, especially when considering a multivariate setup.

7.2 The Environment of the Simulated Car

The datasets used within this study consist of logs from simulations of a car running on flat ground. Only considering a flat ground environment provided a simplified and concise problem to work with. We have through experimentation found that machine learning

may be a viable approach to classify whether a PID-controlled system was affected by soft-errors. A natural next step would be to work with a more broad dataset that consists of different environments and scenarios. This could include environments such as uphill, downhill and transitions between environments. We argue that especially these changes between environments are important to include as we assume they would affect the logs and consequently the generated time-series the most. This broader dataset would as a result be used to train the models in a more dynamic setting that better can be applied in a real-world scenario. We do not expect the intricacy of the classification problem to increase by much if matrix profiles are used. This is based on the assumption that these new logs would consist of similar patterns that would be highlighted by the matrix profile.

7.3 Real-world Applications

In this study, we focus on classifying a time-series of a fixed size rather than classifying each point individually. Hence, if one would want to utilise the trained model in a live PID-controlled system, such as the cruise control of a car, this should be considered. Two intuitive approaches to this implementation are: First, the system could keep a memory of the last x[1] number of logged values e.g. the torque values sent by the controller. Then every time the controller sends a new torque value, the updated time-series is given to the pre-trained model for classification. Second, the system could save intervals of x number of the logged values for the time-series. Then at the end of an interval, pass these to the pre-trained model for classification. With the second approach, each time-series passed to the model would be completely different whereas in the first approach, it is only the last entry in the time-series that would be new when predicting the class labels.

As previously stated, applying the models of this study in a live environment naturally requires some extra steps or considerations. However, there exist other more directly applicable use cases. An example of this could be to use the pre-trained models of this study to filter out samples that include abnormal behaviour caused by soft-errors to avoid unwanted outliers in an unlabelled dataset.

7.4 Classification vs. Detection

Other than classification another learning task for time-series is detection. When applying this to our problem one would be trying to determine which points in the time-series are part of the abnormal behaviour. This is typically done by computing an anomaly score for each point in the time-series and if this score is larger than a certain threshold it is considered an anomaly[43].

With time-series detection, we would be able to discern more precisely where in a time-series abnormal behaviour occurs as well as differentiate between multiple anomalous behaviours. Whereas with classification we make one prediction for the whole time-series. We do not use time-series detection in this study as the models considered [43] are trained and tested on only one continues time-series. As the datasets used in this study are transformed into multiple small time-series, these methods are not directly

[1] x is a fixed number determining the size of the time-series considered

applicable. One approach to transform the dataset into one continuous time-series could be to concatenate the time-series generated from the datasets. Although this is expected to introduce unnatural patterns that would harm the integrity of the problem. Given these considerations, time-series detection methods were discarded.

Conclusion 8

Soft-errors are becoming an increasing concern as chip density grows. Furthermore, current mitigation techniques require additional or specialised hardware. This motivates the search for new and more efficient approaches for mitigating soft-errors. In this study, we have explored the use of machine learning as a potential solution. Here, we used the "LANSCE 18 Cruise" dataset and two simulated extensions to train different machine learning models to classify whether a soft-error has occurred in a PID controller. A variety of machine learning models were tested since this is still a new problem. The models tested are: Logistic Regression (LR), Support Vector Machine (SVM), Simple Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and the two state-of-the-art time-series classification models Random Convolutional Kernel Transform (ROCKET) and HIVE-COTE 2.0 (HC2). We furthermore experimented with transforming the data into matrix profiles to potentially highlight certain features that would benefit the machine learning models' ability to train.

We concluded during the experimentation process that, based on the baseline experiments, more specialised approaches were needed to achieve any meaningful performance. This conclusion is based on the fact that all of these results are similar or worse than random guessing with accuracies of ≤ 0.58. Experiments with the time-series classification models showed a noticeable increase in accuracies. Notably HC2 on the original dataset and ROCKET on the simulated dataset, both using torque as the input feature, yielded accuracies of 0.65 and 0.67 respectively. These are the highest accuracies recorded in experiments without matrix profiles.

Using matrix profiles as input data to the machine learning models showed a general increase in performance. This supports our assumption of matrix profiles being useful to the machine learning models as they highlight anomalous behaviour in the logged data. Noteworthy results here are gained with the SVM and ROCKET models trained on the simulated dataset with torque as the input feature. These experiments reached an accuracy of 0.77 and 0.96 respectively. Furthermore, some experiments with matrix profiles as input to the RNN models also performed exceptionally well. Here the GRU model with the A2 architecture trained on the simulated dataset with torque as the input feature and the matrix profiles $\mathbf{P}_{[15,25]}$ achieved an accuracy of 0.99.

The results achieved in this study suggest that machine learning is a promising approach for detecting abnormal behaviour caused by soft-errors in a PID-controlled environment.

Bibliography

[1] Stephen Buchner and Dale McMorrow. Overview of single event effects. Presentation at SERESSA 2015. URL https://www.inaoep.mx/seressa2015/archivos/Lunes_16_30%20%20Buchner.pdf. Last visited: 2023-05-19.

[2] Soft errors in electronic memory – a white paper, 2004. URL https://tezzaron.com/media/soft_errors_1_1_secure.pdf. Last visited: 2023-05-19.

[3] Simon Kanne Mikkelsen and Anna Veibel Bonde. Representing real-world data to detect soft-errors with machine learning - a study analysing and applying the "lansce 18 cruise" dataset. Private source, 2022.

[4] Sean Kauffman. Lansce 18 cruise documentation. Private source, 2018.

[5] Sean Kauffman. Lansce 18 cruise dataset. Private source, 2018.

[6] Mostafa Mohamed Sayed. Applications of pid controllers in industry, 2023. URL https://www.linkedin.com/pulse/applications-pid-controllers-industry-mostafa-mohamed-sayed. Last visited: 2023-05-19.

[7] Machine learning with applications. *Machine Learning with Applications*, 6:1–2, 2022. ISSN 2666-8270. URL https://www.sciencedirect.com/journal/machine-learning-with-applications.

[8] Mary Pratt. 10 common uses for machine learning applications in business. URL https://www.techtarget.com/searchenterpriseai/feature/10-common-uses-for-machine-learning-applications-in-business. Last visited: 2023-05-19.

[9] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn Keogh. Matrix profile i: All pairs similarity joins for time series: A unifying view that includes motifs, discords and shapelets. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*, pages 1317–1322, 2016. doi: 10.1109/ICDM.2016.0179.

[10] Andrew Van Benschoten, Austin Ouyang, Francisco Bischoff, and Tyler Marrs. Mpa: a novel cross-language api for time series analysis. *Journal of Open Source Software*, 5(49):2179, 2020. doi: 10.21105/joss.02179.

[11] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990. ISSN 0364-0213. doi: https://doi.org/10.1016/0364-0213(90)90002-E. URL https://www.sciencedirect.com/science/article/pii/036402139090002E.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

[13] Greg Van Houdt, Carlos Mosquera, and Gonzalo Nápoles. A review on the long short-term memory model. *Artificial Intelligence Review*, 53, 12 2020. doi: 10.1007/s10462-020-09838-1.

[14] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling, 2014.

[15] Time series classification. URL `https://paperswithcode.com/task/time-series-classification`. Last visited: 2023-05-19.

[16] Alejandro Pasos Ruiz, Michael Flynn, James Large, Matthew Middlehurst, and Anthony Bagnall. The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, 35(2):401–449, Mar 2021. ISSN 1573-756X. doi: 10.1007/s10618-020-00727-3.

[17] Angus Dempster, François Petitjean, and Geoffrey I Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020.

[18] Anthony Bagnall, Jason Lines, Michael Flynn, and James Large. Hive-cote 2.0: a new meta ensemble for time series classification. *Machine Learning*, 110(3):489–514, 2021.

[19] Anthony Bagnall, Hoang Anh Dau, Jason Lines, Michael Flynn, James Large, Aaron Bostrom, Paul Southam, and Eamonn Keogh. The uea multivariate time series classification archive, 2018, 2018.

[20] Tyler Marrs. Introduction to matrix profiles, 2019. URL `https://towardsdatascience.com/introduction-to-matrix-profiles-5568f3375d90`. Last visited: 2023-05-19.

[21] Eamonn J. Keogh, Jessica Lin, and Ada Wai-Chee Fu. Hot sax: Finding the most unusual time series subsequence: Algorithms and applications. 2004.

[22] Anh Dau and Eamonn Keogh. Matrix profile v: A generic technique to incorporate domain knowledge into motif discovery. pages 125–134, 08 2017. doi: 10.1145/3097983.3097993.

[23] Riccardo Guidotti and Matteo D'Onofrio. Matrix profile-based interpretable time series classifier. *Frontiers in Artificial Intelligence*, 4, 2021. ISSN 2624-8212. doi: 10.3389/frai.2021.699448. URL `https://www.frontiersin.org/articles/10.3389/frai.2021.699448`.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[25] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

[26] Vivek Singh. Basic architecture of rnn and lstm, 2017. URL https://pydeeplearning.weebly.com/blog/basic-architecture-of-rnn-and-lstm. Last visited: 2023-05-19.

[27] Mehreen Saeed. An introduction to recurrent neural networks and the math that powers them, 2023. URL https://machinelearningmastery.com/an-introduction-to-recurrent-neural-networks-and-the-math-that-powers-them/. Last visited: 2023-05-19.

[28] Thomas Wood. Backpropagation. URL https://deepai.org/machine-learning-glossary-and-terms/backpropagation. Last visited: 2023-05-19.

[29] Christopher Olah. Understanding lstm networks. URL http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Last visited: 2023-05-19.

[30] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. 2014. doi: 10.48550/arXiv.1406.1078.

[31] Rnn, lstm & gru, 2019. URL http://dprogrammer.org/rnn-lstm-gru. Last visited: 2023-05-19.

[32] Jason Brownlee. How to choose loss functions when trainning deep learning neural networks. URL https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/. Last visited: 2023-05-19.

[33] Sanket Doshi. Optimizers in deep learning, 2019. URL https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6. Last visited: 2023-05-19.

[34] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[35] Jason Lines, Luke Davis, Jon Hills, and Anthony Bagnall. A shapelet transform for time series classification. pages 289–297, 08 2012. doi: 10.1145/2339530.2339579.

[36] Anthony J. Bagnall, Michael Flynn, James Large, Jason Lines, and Matthew Middlehurst. A tale of two toolkits, report the third: on the usage and performance of HIVE-COTE v1.0. *CoRR*, abs/2004.06069, 2020. URL https://arxiv.org/abs/2004.06069.

[37] Matthew Middlehurst, James Large, Gavin C. Cawley, and Anthony J. Bagnall. The temporal dictionary ensemble (TDE) classifier for time series classification. *CoRR*, abs/2105.03841, 2021. URL https://arxiv.org/abs/2105.03841.

[38] Matthew Middlehurst, James Large, and Anthony J. Bagnall. The canonical interval forest (CIF) classifier for time series classification. *CoRR*, abs/2008.09172, 2020. URL `https://arxiv.org/abs/2008.09172`.

[39] Harleen Kaur. Understanding accuracy, recall, precision and f1 scores and confusion matrices. URL `https://towardsdatascience.com/understanding-accuracy-\recall-precision-f1-scores-and-confusion-matrices-561e0f5e328c`. Last visited: 2023-05-19.

[40] François Chollet et al. Keras. `https://keras.io`, 2015.

[41] Aman Sachdev. Choosing number of hidden layers and number of hidden neurons in neural networks, 2018. URL `https://www.linkedin.com/pulse/choosing-number-hidden-layers-neurons-neural-networks-sachdev`. Last visited: 2023-05-19.

[42] Markus Löning, Anthony Bagnall, Sajaysurya Ganesh, Viktor Kazakov, Jason Lines, and Franz J Király. sktime: A unified interface for machine learning with time series. *arXiv preprint arXiv:1909.07872*, 2019.

[43] Siwon Kim, Kukjin Choi, Hyun-Soo Choi, Byunghan Lee, and Sungroh Yoon. Towards a rigorous evaluation of time-series anomaly detection. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(7):7194–7201, June 2022. doi: 10.1609/aaai.v36i7.20680. URL `https://ojs.aaai.org/index.php/AAAI/article/view/20680`.

Abbreviations A

Adam Adaptive Moment Estimation.

Bagging Bootstrap Aggregation.
BPTT backpropagation through time.

DrCIF Diverse Representation Canonical Interval Forest.

ECC Error correcting code.

GRU Gated Recurrent Unit.

HC2 HIVE-COTE 2.0.

LR Logistic Regression.
LSTM Long Short-Term Memory.

NN Neural Network.

RNN Recurrent Neural Network.
ROCKET Random Convolutional Kernel Transform.

SEE Single Event Error.
STC Shapelet Transform Classifier.
SVM Support Vector Machine.

TDE Temporal Dictionary Ensemble.

Experimental-type Runs with Anomalies

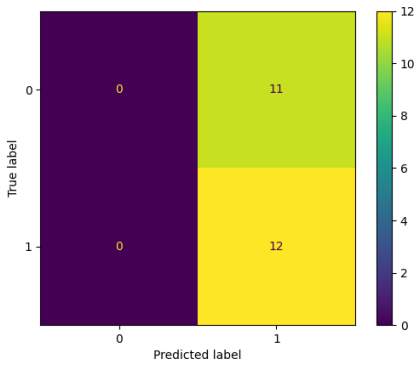
B

The run IDs of the 56 experimental-type runs that deviates in behaviour from their corresponding nominal-type run.

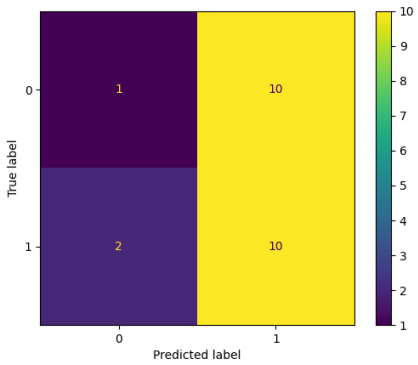
kp105ki8kd1sp20experiment54
kp116ki3kd2sp18experiment105
kp122ki13kd1sp13experiment29
kp122ki13kd1sp13experiment50
kp131ki11kd2sp14experiment37
kp131ki11kd2sp14experiment89
kp149ki11kd3sp18experiment82
kp151ki10kd3sp2experiment59
kp151ki10kd3sp2experiment60
kp151ki10kd3sp2experiment94
kp188ki3kd3sp18experiment39
kp238ki3kd2sp5experiment51
kp260ki7kd1sp11experiment46
kp260ki7kd1sp11experiment47
kp261ki2kd3sp4experiment84
kp270ki3kd3sp15experiment128
kp274ki9kd3sp15experiment80
kp293ki9kd1sp2experiment23
kp293ki9kd1sp2experiment79
kp297ki4kd3sp5experiment18
kp332ki14kd1sp12experiment69
kp336ki10kd3sp6experiment117
kp336ki10kd3sp6experiment25
kp346ki3kd2sp9experiment129
kp346ki3kd2sp9experiment42
kp374ki4kd2sp9experiment67
kp494ki15kd3sp1experiment44
kp494ki15kd3sp1experiment63
kp544ki5kd2sp16experiment36
kp544ki5kd2sp16experiment81
kp583ki15kd1sp0experiment126
kp583ki15kd1sp0experiment37
kp611ki3kd1sp0experiment71
kp621ki4kd1sp13experiment30
kp634ki6kd2sp11experiment51
kp654ki10kd3sp7experiment129
kp654ki10kd3sp7experiment23
kp654ki10kd3sp7experiment25
kp695ki5kd2sp13experiment21
kp718ki5kd1sp16experiment113
kp718ki5kd1sp16experiment17
kp729ki4kd1sp5experiment31
kp731ki10kd3sp3experiment86
kp746ki3kd1sp4experiment29
kp746ki3kd1sp4experiment83
kp770ki2kd1sp20experiment44
kp770ki2kd1sp20experiment76
kp771ki15kd3sp15experiment53
kp790ki9kd1sp6experiment37
kp790ki9kd1sp6experiment62
kp790ki9kd1sp6experiment65
kp831ki11kd3sp18experiment43
kp840ki8kd2sp17experiment18
kp840ki8kd2sp17experiment67
kp952ki3kd1sp12experiment16
kp972ki9kd2sp3experiment90

Confusion Matrices C

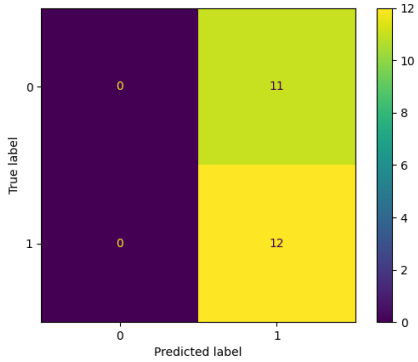
The confusion matrices from the two 'Baseline' experiments with the SVM model trained on the original and simulated dataset respectively both with torque as the input feature. The resulting confusion matrices of the three individual run of each of the two experiments can be found in the two figures figures C.1 and C.2.



(a) Confusion matrix from the first run.

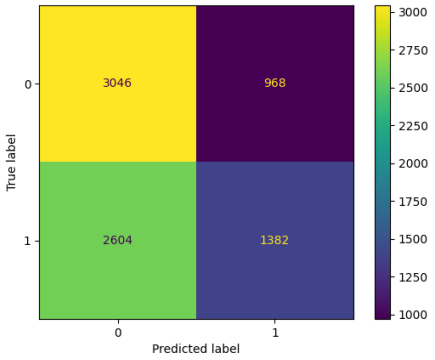


(b) Confusion matrix from the second run.

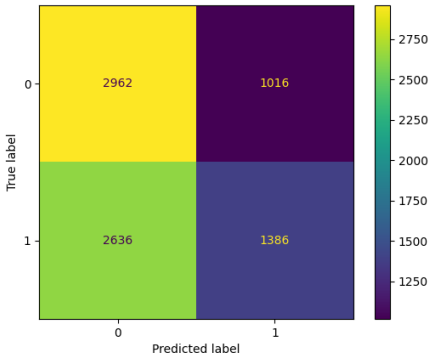


(c) Confusion matrix from the third run.

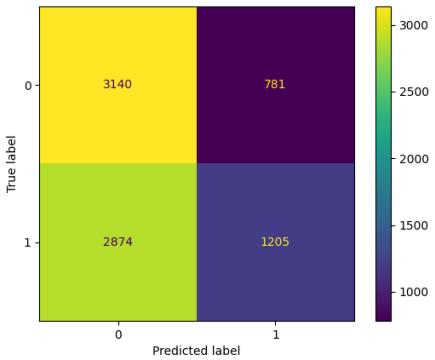
Figure C.1. Confusion matrices for all tree runs where an SVM model was trained on the original dataset with torque as the input feature.



(a) Confusion matrix from the first run.



(b) Confusion matrix from the second run.



(c) Confusion matrix from the third run.

Figure C.2. Confusion matrices for all tree runs where an SVM model was trained on the simulated dataset with torque as the input feature.

Experiment Results D

This appendix lists all results related to the aggregated results found in Section 6.3. Each experiment is as stated in Section 6.1 run three individual times where the only difference is the seed used for data shuffling and weight initialisation. The values presented in Section 6.3 are the aggregated arithmetic mean values and standard deviation listed in parenthesis. Dataset and input feature specifications are described in Section 6.1.

D.1 Baseline

Logistic Regression

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.52	0.52	0.52
2	0.43	0.43	0.43	0.43
3	0.35	0.34	0.34	0.34
Mean	0.43 (0.09)	0.43 (0.09)	0.43 (0.09)	0.43 (0.09)

Table D.1. LR trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.39	0.39	0.39
2	0.39	0.36	0.38	0.36
3	0.35	0.34	0.34	0.34
Mean	0.38 (0.02)	0.36 (0.03)	0.37 (0.03)	0.36 (0.03)

Table D.2. LR trained on the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.51	0.51	0.51	0.51
2	0.52	0.52	0.52	0.51
3	0.52	0.52	0.52	0.51
Mean	0.52 (0.01)	0.52 (0.01)	0.52 (0.01)	0.51 (0.0)

Table D.3. LR trained on the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.52	0.52	0.52
2	0.52	0.52	0.52	0.52
3	0.52	0.52	0.52	0.52
Mean	0.52 (0.0)	0.52 (0.0)	0.52 (0.0)	0.52 (0.0)

Table D.4. LR trained on the simulated dataset with torque as the input feature.

Support Vector Machine

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.26	0.5	0.34
2	0.52	0.26	0.5	0.34
3	0.52	0.26	0.5	0.34
Mean	0.52 (0.0)	0.26 (0.0)	0.5 (0.0)	0.34 (0.0)

Table D.5. SVM trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.26	0.5	0.34
2	0.48	0.42	0.46	0.38
3	0.52	0.26	0.5	0.34
Mean	0.51 (0.02)	0.31 (0.09)	0.49 (0.02)	0.35 (0.02)

Table D.6. SVM trained on the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.53	0.53	0.53	0.52
2	0.53	0.53	0.53	0.53
3	0.52	0.55	0.52	0.44
Mean	0.53 (0.01)	0.54 (0.01)	0.53 (0.01)	0.5 (0.05)

Table D.7. SVM trained on the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.55	0.56	0.55	0.53
2	0.54	0.55	0.54	0.53
3	0.54	0.56	0.55	0.51
Mean	0.54 (0.01)	0.56 (0.01)	0.55 (0.01)	0.52 (0.01)

Table D.8. SVM trained on the simulated dataset with torque as the input feature.

D.2 RNN

The names A1, A2, and A3 used in this section refer to the architecture of the RNN models specified in Section 6.2. The experiment runs marked with '*' encountered a 'Not a Number' error which results in the model stopping learning prematurely.

SimpleRNN

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.26	0.5	0.34
2	0.52	0.26	0.5	0.34
3	0.52	0.26	0.5	0.34
Mean	0.52 (0.0)	0.26 (0.0)	0.5 (0.0)	0.34 (0.0)

Table D.9. SimpleRNN (A1) trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.26	0.5	0.34
2	0.52	0.26	0.5	0.34
3	0.52	0.26	0.5	0.34
Mean	0.52 (0.0)	0.26 (0.0)	0.5 (0.0)	0.34 (0.0)

Table D.10. SimpleRNN (A2) trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.26	0.5	0.34
2	0.52	0.26	0.5	0.34
3	0.52	0.26	0.5	0.34
Mean	0.52 (0.0)	0.26 (0.0)	0.5 (0.0)	0.34 (0.0)

Table D.11. SimpleRNN (A3) trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.57	0.76	0.58	0.49
2	0.48	0.49	0.5	0.38
3	0.57	0.76	0.58	0.49
Mean	0.54 (0.05)	0.67 (0.16)	0.55 (0.05)	0.45 (0.06)

Table D.12. SimpleRNN (A1) trained on the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.57	0.56	0.56	0.56
2	0.52	0.51	0.51	0.46
3	0.57	0.61	0.55	0.49
Mean	0.55 (0.03)	0.56 (0.05)	0.54 (0.03)	0.5 (0.05)

Table D.13. SimpleRNN (A2) trained on the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.51	0.51	0.49
2	0.52	0.52	0.52	0.52
3	0.57	0.58	0.55	0.52
Mean	0.54 (0.03)	0.54 (0.04)	0.53 (0.02)	0.51 (0.02)

Table D.14. SimpleRNN (A3) trained on the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.55	0.53	0.49
2	0.48	0.49	0.49	0.42
3	0.57	0.77	0.55	0.44
Mean	0.52 (0.05)	0.6 (0.15)	0.52 (0.03)	0.45 (0.04)

Table D.15. SimpleRNN (A1) trained on the original dataset with vehicle speed and torque as the input features.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.36	0.42	0.36
2	0.57	0.6	0.58	0.54
3	0.52	0.51	0.51	0.46
Mean	0.51 (0.07)	0.49 (0.12)	0.5 (0.08)	0.45 (0.09)

Table D.16. SimpleRNN (A2) trained on the original dataset with vehicle speed and torque as the input features.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.39	0.4	0.38
2	0.48	0.48	0.48	0.47
3	0.57	0.64	0.58	0.52
Mean	0.48 (0.09)	0.5 (0.13)	0.49 (0.09)	0.46 (0.07)

Table D.17. SimpleRNN (A3) trained on the original dataset with vehicle speed and torque as the input features.

Long Short-Term Memory

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.48	0.49	0.49	0.45
2	0.43	0.44	0.44	0.43
3	0.52	0.26	0.5	0.34
Mean	0.48 (0.05)	0.4 (0.12)	0.48 (0.03)	0.41 (0.06)

Table D.18. LSTM (A1) trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.42	0.45	0.39
2	0.48	0.48	0.48	0.48
3	0.35	0.2	0.36	0.26
Mean	0.42 (0.07)	0.37 (0.15)	0.43 (0.06)	0.38 (0.11)

Table D.19. LSTM (A2) trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.34	0.41	0.33
2	0.48	0.48	0.48	0.48
3	0.35	0.2	0.36	0.26
Mean	0.41 (0.07)	0.34 (0.14)	0.42 (0.06)	0.36 (0.11)

Table D.20. LSTM (A3) trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.57	0.58	0.57	0.56
2	0.43	0.39	0.45	0.36
3	0.52	0.53	0.53	0.52
Mean	0.51 (0.07)	0.5 (0.1)	0.52 (0.06)	0.48 (0.11)

Table D.21. LSTM (A1) trained on the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.34	0.41	0.33
2	0.48	0.48	0.48	0.47
3	0.65	0.68	0.66	0.65
Mean	0.51 (0.13)	0.5 (0.17)	0.52 (0.13)	0.48 (0.16)

Table D.22. LSTM (A2) trained on the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.37	0.4	0.36
2	0.61	0.61	0.61	0.61
3	0.57	0.61	0.55	0.49
Mean	0.52 (0.12)	0.53 (0.14)	0.52 (0.11)	0.49 (0.13)

Table D.23. LSTM (A3) trained on the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.65	0.68	0.66	0.65
2	0.48	0.49	0.49	0.45
3	0.61	0.64	0.62	0.6
Mean	0.58 (0.09)	0.6 (0.1)	0.59 (0.09)	0.57 (0.1)

Table D.24. LSTM (A1) trained on the original dataset with vehicle speed and torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.48	0.47	0.47	0.47
2	0.48	0.49	0.49	0.45
3	0.48	0.42	0.46	0.38
Mean	0.48 (0.0)	0.46 (0.04)	0.47 (0.02)	0.43 (0.05)

Table D.25. LSTM (A2) trained on the original dataset with vehicle speed and torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.37	0.4	0.36
2	0.39	0.39	0.4	0.38
3	0.43	0.43	0.44	0.42
Mean	0.4 (0.02)	0.4 (0.03)	0.41 (0.02)	0.39 (0.03)

Table D.26. LSTM (A3) trained on the original dataset with vehicle speed and torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.25	0.5	0.33
2	0.5	0.25	0.5	0.33
3	0.51	0.51	0.5	0.36
Mean	0.5 (0.01)	0.34 (0.15)	0.5 (0.0)	0.34 (0.02)

Table D.27. LSTM (A1) trained on the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.5	0.5	0.37
2	0.5	0.25	0.5	0.33
3	0.49	0.25	0.5	0.33
Mean	0.5 (0.01)	0.33 (0.14)	0.5 (0.0)	0.34 (0.02)

Table D.28. LSTM (A2) trained on the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.5	0.5	0.37
2	0.5	0.25	0.5	0.33
3	0.49	0.25	0.5	0.33
Mean	0.5 (0.01)	0.33 (0.14)	0.5 (0.0)	0.34 (0.02)

Table D.29. LSTM (A3) trained on the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.51	0.5	0.4
2	0.51	0.51	0.51	0.46
3	0.5	0.53	0.51	0.39
Mean	0.5 (0.01)	0.52 (0.01)	0.51 (0.01)	0.42 (0.04)

Table D.30. LSTM (A1) trained on the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.51	0.51	0.51	0.5
2	0.51	0.51	0.51	0.48
3	0.51	0.5	0.5	0.48
Mean	0.51 (0.0)	0.51 (0.01)	0.51 (0.01)	0.49 (0.01)

Table D.31. LSTM (A2) trained on the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.51	0.51	0.51	0.5
2	0.51	0.51	0.51	0.49
3*	0.49	0.25	0.5	0.33
Mean	0.5 (0.01)	0.42 (0.15)	0.51 (0.01)	0.44 (0.1)

Table D.32. LSTM (A3) trained on the simulated dataset with torque as the input feature.

Gated Recurrent Unit

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.57	0.58	0.57	0.56
2	0.48	0.48	0.48	0.48
3	0.35	0.2	0.36	0.26
Mean	0.47 (0.11)	0.42 (0.2)	0.47 (0.11)	0.43 (0.16)

Table D.33. GRU (A1) trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.42	0.45	0.39
2	0.48	0.48	0.48	0.48
3	0.35	0.35	0.35	0.35
Mean	0.42 (0.07)	0.42 (0.07)	0.43 (0.07)	0.41 (0.07)

Table D.34. GRU (A2) trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.34	0.41	0.33
2	0.48	0.48	0.48	0.48
3	0.39	0.32	0.38	0.33
Mean	0.42 (0.05)	0.38 (0.09)	0.42 (0.05)	0.38 (0.09)

Table D.35. GRU (A3) trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.57	0.56	0.56	0.56
2	0.57	0.57	0.56	0.54
3	0.57	0.57	0.57	0.56
Mean	0.57 (0.0)	0.57 (0.01)	0.56 (0.01)	0.55 (0.01)

Table D.36. GRU (A1) trained on the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.58	0.54	0.46
2	0.57	0.57	0.56	0.54
3	0.57	0.77	0.55	0.44
Mean	0.55 (0.03)	0.64 (0.11)	0.55 (0.01)	0.48 (0.05)

Table D.37. GRU (A2) trained on the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.48	0.49	0.5	0.38
2	0.48	0.42	0.46	0.38
3	0.61	0.79	0.59	0.52
Mean	0.52 (0.08)	0.57 (0.2)	0.52 (0.07)	0.43 (0.08)

Table D.38. GRU (A3) trained on the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.57	0.6	0.58	0.54
2	0.43	0.44	0.44	0.43
3	0.65	0.71	0.66	0.63
Mean	0.55 (0.11)	0.58 (0.14)	0.56 (0.11)	0.53 (0.1)

Table D.39. GRU (A1) trained on the original dataset with vehicle speed and torque as the input features.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.42	0.45	0.39
2	0.48	0.49	0.49	0.45
3	0.48	0.45	0.47	0.43
Mean	0.46 (0.03)	0.45 (0.04)	0.47 (0.02)	0.42 (0.03)

Table D.40. GRU (A2) trained on the original dataset with vehicle speed and torque as the input features.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.35	0.33	0.36	0.33
2	0.39	0.39	0.39	0.39
3	0.52	0.51	0.51	0.46
Mean	0.42 (0.09)	0.41 (0.09)	0.42 (0.08)	0.39 (0.07)

Table D.41. GRU (A3) trained on the original dataset with vehicle speed and torque as the input features.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.5	0.5	0.37
2	0.5	0.25	0.5	0.33
3	0.51	0.51	0.5	0.35
Mean	0.5 (0.01)	0.42 (0.15)	0.5 (0.0)	0.35 (0.02)

Table D.42. GRU (A1) trained on the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.5	0.5	0.37
2	0.5	0.25	0.5	0.33
3	0.51	0.51	0.5	0.34
Mean	0.5 (0.01)	0.42 (0.15)	0.5 (0.0)	0.35 (0.02)

Table D.43. GRU (A2) trained on the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.25	0.5	0.33
2	0.5	0.25	0.5	0.33
3	0.51	0.25	0.5	0.34
Mean	0.5 (0.01)	0.25 (0.0)	0.5 (0.0)	0.33 (0.01)

Table D.44. GRU (A3) trained on the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.51	0.5	0.34
2	0.5	0.5	0.5	0.49
3	0.51	0.59	0.5	0.34
Mean	0.5 (0.01)	0.53 (0.05)	0.5 (0.0)	0.39 (0.09)

Table D.45. GRU (A1) trained on the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.51	0.5	0.36
2	0.5	0.51	0.5	0.44
3	0.52	0.52	0.51	0.44
Mean	0.51 (0.01)	0.51 (0.01)	0.5 (0.01)	0.41 (0.05)

Table D.46. GRU (A2) trained on the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.25	0.5	0.33
2	0.5	0.55	0.5	0.36
3	0.51	0.52	0.5	0.35
Mean	0.5 (0.01)	0.44 (0.17)	0.5 (0.0)	0.35 (0.02)

Table D.47. GRU (A3) trained on the simulated dataset with torque as the input feature.

D.3 Time-series Classification

ROCKET

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.57	0.58	0.55	0.52
2	0.48	0.45	0.47	0.43
3	0.52	0.52	0.52	0.52
Mean	0.52 (0.05)	0.52 (0.07)	0.51 (0.04)	0.49 (0.05)

Table D.48. ROCKET trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.38	0.39	0.38
2	0.39	0.36	0.38	0.36
3	0.26	0.26	0.26	0.26
Mean	0.35 (0.08)	0.33 (0.06)	0.34 (0.07)	0.33 (0.06)

Table D.49. ROCKET trained on the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.64	0.64	0.64	0.64
2	0.64	0.64	0.64	0.63
3	0.63	0.63	0.63	0.63
Mean	0.64 (0.01)	0.64 (0.01)	0.64 (0.01)	0.63 (0.01)

Table D.50. ROCKET trained on the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.67	0.67	0.67	0.66
2	0.67	0.67	0.67	0.66
3	0.66	0.67	0.66	0.66
Mean	0.67 (0.01)	0.67 (0.0)	0.67 (0.01)	0.66 (0.0)

Table D.51. ROCKET trained on the simulated dataset with torque as the input feature.

HIVE-COTE 2.0

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.35	0.32	0.34	0.33
2	0.3	0.3	0.3	0.3
3	0.35	0.35	0.35	0.35
Mean	0.33 (0.03)	0.32 (0.03)	0.33 (0.03)	0.33 (0.03)

Table D.52. HC2 trained on the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.65	0.71	0.64	0.62
2	0.52	0.52	0.52	0.52
3	0.78	0.85	0.77	0.77
Mean	0.65 (0.13)	0.69 (0.17)	0.64 (0.13)	0.64 (0.13)

Table D.53. HC2 trained on the original dataset with torque as the input feature.

D.4 Matrix Profile for non-RNN models

Matrix profile experiments and window size notation is described in Section 6.2.

Logistic Regression

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.3	0.29	0.3	0.29
2	0.39	0.39	0.39	0.39
3	0.48	0.47	0.47	0.47
Mean	0.39 (0.09)	0.38 (0.09)	0.39 (0.09)	0.38 (0.09)

Table D.54. LR trained on the matrix profile $\mathbf{P}_5$ created from the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.39	0.39	0.39
2	0.48	0.48	0.48	0.48
3	0.35	0.34	0.34	0.34
Mean	0.41 (0.07)	0.4 (0.07)	0.4 (0.07)	0.4 (0.07)

Table D.55. LR trained on the matrix profile $\mathbf{P}_{15}$ created from the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.3	0.3	0.3	0.3
2	0.35	0.34	0.34	0.34
3	0.3	0.3	0.3	0.3
Mean	0.32 (0.03)	0.31 (0.02)	0.31 (0.02)	0.31 (0.02)

Table D.56. LR trained on the matrix profile $\mathbf{P}_{25}$ created from the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.48	0.48	0.48	0.48
2	0.57	0.57	0.57	0.56
3	0.43	0.4	0.42	0.39
Mean	0.49 (0.07)	0.48 (0.09)	0.49 (0.08)	0.48 (0.09)

Table D.57. LR trained on the matrix profile $\mathbf{P}_{100}$ created from the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.3	0.26	0.31	0.27
2	0.39	0.39	0.39	0.39
3	0.35	0.35	0.35	0.34
Mean	0.35 (0.05)	0.33 (0.07)	0.35 (0.04)	0.33 (0.06)

Table D.58. LR trained on the matrix profile $\mathbf{P}_5$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.35	0.33	0.36	0.33
2	0.35	0.35	0.35	0.35
3	0.48	0.48	0.48	0.47
Mean	0.39 (0.08)	0.39 (0.08)	0.4 (0.07)	0.38 (0.08)

Table D.59. LR trained on the matrix profile $\mathbf{P}_{15}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.3	0.26	0.31	0.27
2	0.43	0.44	0.44	0.43
3	0.57	0.56	0.56	0.56
Mean	0.43 (0.14)	0.42 (0.15)	0.44 (0.13)	0.42 (0.15)

Table D.60. LR trained on the matrix profile $\mathbf{P}_{25}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.42	0.43	0.42
2	0.43	0.43	0.43	0.43
3	0.39	0.38	0.39	0.38
Mean	0.42 (0.02)	0.41 (0.03)	0.42 (0.02)	0.41 (0.03)

Table D.61. LR trained on the matrix profile $\mathbf{P}_{100}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.5	0.5	0.5
2	0.51	0.51	0.51	0.51
3	0.5	0.5	0.5	0.5
Mean	0.5 (0.01)	0.5 (0.01)	0.5 (0.01)	0.5 (0.01)

Table D.62. LR trained on the matrix profile $\mathbf{P}_5$ created from the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.5	0.5	0.5
2	0.51	0.51	0.51	0.51
3	0.51	0.52	0.52	0.51
Mean	0.51 (0.01)	0.51 (0.01)	0.51 (0.01)	0.51 (0.01)

Table D.63. LR trained on the matrix profile $\mathbf{P}_{15}$ created from the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.51	0.51	0.51	0.51
2	0.51	0.51	0.51	0.51
3	0.51	0.51	0.51	0.51
Mean	0.51 (0.0)	0.51 (0.0)	0.51 (0.0)	0.51 (0.0)

Table D.64. LR trained on the matrix profile $\mathbf{P}_{25}$ created from the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.51	0.51	0.51	0.51
2	0.51	0.51	0.51	0.51
3	0.51	0.51	0.51	0.51
Mean	0.51 (0.0)	0.51 (0.0)	0.51 (0.0)	0.51 (0.0)

Table D.65. LR trained on the matrix profile $\mathbf{P}_{100}$ created from the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.63	0.64	0.63	0.63
2	0.62	0.63	0.62	0.62
3	0.63	0.63	0.63	0.63
Mean	0.63 (0.01)	0.63 (0.01)	0.63 (0.01)	0.63 (0.01)

Table D.66. LR trained on the matrix profile $\mathbf{P}_{5}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.66	0.67	0.66	0.66
2	0.66	0.66	0.66	0.66
3	0.66	0.67	0.66	0.66
Mean	0.66 (0.0)	0.67 (0.01)	0.66 (0.0)	0.66 (0.0)

Table D.67. LR trained on the matrix profile $\mathbf{P}_{15}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.66	0.66	0.66	0.66
2	0.66	0.67	0.66	0.66
3	0.66	0.67	0.66	0.66
Mean	0.66 (0.0)	0.67 (0.01)	0.66 (0.0)	0.66 (0.0)

Table D.68. LR trained on the matrix profile $\mathbf{P}_{25}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.54	0.54	0.54	0.54
2	0.54	0.54	0.54	0.54
3	0.54	0.54	0.54	0.54
Mean	0.54 (0.0)	0.54 (0.0)	0.54 (0.0)	0.54 (0.0)

Table D.69. LR trained on the matrix profile $\mathbf{P}_{100}$ created from the simulated dataset with torque as the input feature.

Support Vector Machine

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.36	0.42	0.36
2	0.48	0.25	0.46	0.32
3	0.48	0.25	0.46	0.32
Mean	0.46 (0.03)	0.29 (0.06)	0.45 (0.02)	0.33 (0.02)

Table D.70. SVM trained on the matrix profile $\mathbf{P}_{5}$ created from the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.24	0.42	0.3
2	0.52	0.26	0.5	0.34
3	0.43	0.24	0.42	0.3
Mean	0.46 (0.05)	0.25 (0.01)	0.45 (0.05)	0.31 (0.02)

Table D.71. SVM trained on the matrix profile $\mathbf{P}_{15}$ created from the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.24	0.42	0.3
2	0.43	0.24	0.42	0.3
3	0.52	0.26	0.5	0.34
Mean	0.46 (0.05)	0.25 (0.01)	0.45 (0.05)	0.31 (0.02)

Table D.72. SVM trained on the matrix profile $\mathbf{P}_{25}$ created from the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.43	0.44	0.42
2	0.43	0.36	0.42	0.36
3	0.52	0.55	0.53	0.49
Mean	0.46 (0.05)	0.45 (0.1)	0.46 (0.06)	0.42 (0.07)

Table D.73. SVM trained on the matrix profile $\mathbf{P}_{100}$ created from the original dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.35	0.35	0.35	0.34
2	0.52	0.51	0.51	0.49
3	0.48	0.48	0.48	0.48
Mean	0.45 (0.09)	0.45 (0.09)	0.45 (0.09)	0.44 (0.08)

Table D.74. SVM trained on the matrix profile $\mathbf{P}_{5}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.52	0.52	0.52
2	0.52	0.51	0.51	0.49
3	0.65	0.65	0.65	0.65
Mean	0.56 (0.08)	0.56 (0.08)	0.56 (0.08)	0.55 (0.09)

Table D.75. SVM trained on the matrix profile $\mathbf{P}_{15}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.52	0.52	0.52
2	0.52	0.51	0.51	0.49
3	0.61	0.61	0.61	0.61
Mean	0.55 (0.05)	0.55 (0.06)	0.55 (0.06)	0.54 (0.06)

Table D.76. SVM trained on the matrix profile $\mathbf{P}_{25}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.52	0.52	0.52
2	0.61	0.63	0.6	0.58
3	0.57	0.57	0.57	0.56
Mean	0.57 (0.05)	0.57 (0.06)	0.56 (0.04)	0.55 (0.03)

Table D.77. SVM trained on the matrix profile $\mathbf{P}_{100}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.51	0.51	0.51	0.5
2	0.52	0.52	0.52	0.51
3	0.52	0.52	0.52	0.5
Mean	0.52 (0.01)	0.52 (0.01)	0.52 (0.01)	0.5 (0.01)

Table D.78. SVM trained on the matrix profile $\mathbf{P}_5$ created from the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.51	0.51	0.51	0.51
2	0.52	0.52	0.52	0.51
3	0.52	0.52	0.52	0.5
Mean	0.52 (0.01)	0.52 (0.01)	0.52 (0.01)	0.51 (0.01)

Table D.79. SVM trained on the matrix profile $\mathbf{P}_{15}$ created from the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.51	0.51	0.51	0.49
2	0.52	0.52	0.52	0.49
3	0.51	0.52	0.51	0.48
Mean	0.51 (0.01)	0.52 (0.01)	0.51 (0.01)	0.49 (0.01)

Table D.80. SVM trained on the matrix profile $\mathbf{P}_{25}$ created from the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.55	0.57	0.55	0.52
2	0.54	0.55	0.54	0.51
3	0.53	0.55	0.54	0.51
Mean	0.54 (0.01)	0.56 (0.01)	0.54 (0.01)	0.51 (0.01)

Table D.81. SVM trained on the matrix profile $\mathbf{P}_{100}$ created from the simulated dataset with vehicle speed as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.75	0.75	0.75	0.75
2	0.74	0.74	0.74	0.74
3	0.75	0.75	0.75	0.75
Mean	0.75 (0.01)	0.75 (0.01)	0.75 (0.01)	0.75 (0.01)

Table D.82. SVM trained on the matrix profile $\mathbf{P}_5$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.77	0.77	0.77	0.77
2	0.77	0.77	0.77	0.77
3	0.77	0.77	0.77	0.77
Mean	0.77 (0.0)	0.77 (0.0)	0.77 (0.0)	0.77 (0.0)

Table D.83. SVM trained on the matrix profile $\mathbf{P}_{15}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.75	0.75	0.75	0.75
2	0.75	0.75	0.75	0.75
3	0.75	0.75	0.75	0.75
Mean	0.75 (0.0)	0.75 (0.0)	0.75 (0.0)	0.75 (0.0)

Table D.84. SVM trained on the matrix profile $\mathbf{P}_{25}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.66	0.67	0.66	0.66
2	0.65	0.66	0.65	0.65
3	0.65	0.65	0.65	0.65
Mean	0.65 (0.01)	0.66 (0.01)	0.65 (0.01)	0.65 (0.01)

Table D.85. SVM trained on the matrix profile $\mathbf{P}_{100}$ created from the simulated dataset with torque as the input feature.

ROCKET

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.53	0.53	0.52
2	0.65	0.68	0.66	0.65
3	0.74	0.74	0.74	0.74
Mean	0.64 (0.11)	0.65 (0.11)	0.64 (0.11)	0.64 (0.11)

Table D.86. ROCKET trained on the matrix profile $\mathbf{P}_{5}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.57	0.57	0.57	0.56
2	0.43	0.42	0.45	0.39
3	0.61	0.61	0.61	0.61
Mean	0.54 (0.09)	0.53 (0.1)	0.54 (0.08)	0.52 (0.12)

Table D.87. ROCKET trained on the matrix profile $\mathbf{P}_{15}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.61	0.62	0.61	0.61
2	0.7	0.74	0.7	0.69
3	0.65	0.65	0.65	0.65
Mean	0.65 (0.05)	0.67 (0.06)	0.65 (0.05)	0.65 (0.04)

Table D.88. ROCKET trained on the matrix profile $\mathbf{P}_{25}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.3	0.3	0.3	0.3
2	0.39	0.39	0.39	0.39
3	0.43	0.36	0.42	0.36
Mean	0.37 (0.07)	0.35 (0.05)	0.37 (0.06)	0.35 (0.05)

Table D.89. ROCKET trained on the matrix profile $\mathbf{P}_{100}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.89	0.9	0.89	0.89
2	0.89	0.9	0.89	0.89
3	0.9	0.9	0.9	0.9
Mean	0.89 (0.01)	0.9 (0.0)	0.89 (0.01)	0.89 (0.01)

Table D.90. ROCKET trained on the matrix profile $\mathbf{P}_{5}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.88	0.89	0.88	0.88
2	0.88	0.88	0.88	0.88
3	0.88	0.89	0.88	0.88
Mean	0.88 (0.0)	0.89 (0.01)	0.88 (0.0)	0.88 (0.0)

Table D.91. ROCKET trained on the matrix profile $\mathbf{P}_{15}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.96	0.96	0.96	0.96
2	0.96	0.96	0.96	0.96
3	0.95	0.95	0.95	0.95
Mean	0.96 (0.01)	0.96 (0.01)	0.96 (0.01)	0.96 (0.01)

Table D.92. ROCKET trained on the matrix profile $\mathbf{P}_{25}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.77	0.78	0.77	0.77
2	0.77	0.77	0.77	0.77
3	0.77	0.77	0.77	0.76
Mean	0.77 (0.0)	0.77 (0.01)	0.77 (0.0)	0.77 (0.01)

Table D.93. ROCKET trained on the matrix profile $\mathbf{P}_{100}$ created from the simulated dataset with torque as the input feature.

D.5 Matrix Profile for RNN models

Matrix profile experiments together with architecture and window size notation are described in Section 6.2. Furthermore, the names A1, A2, and A3 used in this section refer to the architecture of the RNN models specified in Section 6.2. The experiment runs marked with '*' encountered a 'Not a Number' error which results in the model stopping learning prematurely.

Long Short-Term Memory

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.23	0.45	0.3
2	0.52	0.26	0.5	0.34
3	0.39	0.21	0.41	0.28
Mean	0.45 (0.07)	0.23 (0.03)	0.45 (0.05)	0.31 (0.03)

Table D.94. LSTM (A1) trained on the matrix profile $\mathbf{P}_{15}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.58	0.54	0.46
2	0.48	0.24	0.5	0.32
3	0.43	0.23	0.45	0.3
Mean	0.48 (0.05)	0.35 (0.2)	0.5 (0.05)	0.36 (0.09)

Table D.95. LSTM (A2) trained on the matrix profile $\mathbf{P}_{15}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.26	0.5	0.34
2	0.52	0.26	0.5	0.34
3	0.39	0.21	0.41	0.28
Mean	0.48 (0.08)	0.24 (0.03)	0.47 (0.05)	0.32 (0.03)

Table D.96. LSTM (A3) trained on the matrix profile $\mathbf{P}_{15}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.48	0.49	0.5	0.38
2	0.52	0.75	0.54	0.41
3	0.43	0.23	0.45	0.3
Mean	0.48 (0.05)	0.49 (0.26)	0.5 (0.05)	0.36 (0.06)

Table D.97. LSTM (A1) trained on the matrix profile $\mathbf{P}_{25}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.43	0.43	0.43
2	0.52	0.26	0.5	0.34
3	0.35	0.2	0.36	0.26
Mean	0.43 (0.09)	0.3 (0.12)	0.43 (0.07)	0.34 (0.09)

Table D.98. LSTM (A2) trained on the matrix profile $\mathbf{P}_{25}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.26	0.5	0.34
2	0.52	0.75	0.54	0.41
3	0.43	0.39	0.45	0.36
Mean	0.49 (0.05)	0.47 (0.25)	0.5 (0.05)	0.37 (0.04)

Table D.99. LSTM (A3) trained on the matrix profile $\mathbf{P}_{25}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.39	0.45	0.36
2	0.7	0.81	0.71	0.67
3	0.52	0.51	0.5	0.41
Mean	0.55 (0.14)	0.57 (0.22)	0.55 (0.14)	0.48 (0.17)

Table D.100. LSTM (A1) trained on the matrix profile $\mathbf{P}_{[15,25]}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.43	0.23	0.45	0.3
2	0.48	0.24	0.5	0.32
3	0.52	0.51	0.5	0.41
Mean	0.48 (0.05)	0.33 (0.16)	0.48 (0.03)	0.34 (0.06)

Table D.101. LSTM (A2) trained on the matrix profile $\mathbf{P}_{[15,25]}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.26	0.5	0.34
2	0.52	0.26	0.5	0.34
3	0.39	0.21	0.41	0.28
Mean	0.48 (0.08)	0.24 (0.03)	0.47 (0.05)	0.32 (0.03)

Table D.102. LSTM (A3) trained on the matrix profile $\mathbf{P}_{[15,25]}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.59	0.76	0.59	0.51
2*	0.5	0.25	0.5	0.33
3	0.51	0.49	0.5	0.35
Mean	0.53 (0.05)	0.5 (0.26)	0.53 (0.05)	0.4 (0.1)

Table D.103. LSTM (A1) trained on the matrix profile $\mathbf{P}_{15}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.25	0.5	0.33
2	0.5	0.25	0.5	0.33
3	0.51	0.55	0.52	0.43
Mean	0.5 (0.01)	0.35 (0.17)	0.51 (0.01)	0.36 (0.06)

Table D.104. LSTM (A2) trained on the matrix profile $\mathbf{P}_{15}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.25	0.5	0.33
2	0.5	0.25	0.5	0.33
3	0.49	0.25	0.5	0.33
Mean	0.5 (0.01)	0.25 (0.0)	0.5 (0.0)	0.33 (0.0)

Table D.105. LSTM (A3) trained on the matrix profile $\mathbf{P}_{15}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.45	0.44	0.45	0.43
2	0.67	0.7	0.67	0.66
3	0.51	0.48	0.5	0.35
Mean	0.54 (0.11)	0.54 (0.14)	0.54 (0.12)	0.48 (0.16)

Table D.106. LSTM (A1) trained on the matrix profile $\mathbf{P}_{25}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.25	0.5	0.33
2	0.88	0.9	0.88	0.88
3*	0.49	0.25	0.5	0.33
Mean	0.62 (0.22)	0.47 (0.38)	0.63 (0.22)	0.51 (0.32)

Table D.107. LSTM (A2) trained on the matrix profile $\mathbf{P}_{25}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.25	0.5	0.33
2	0.5	0.25	0.5	0.33
3	0.49	0.25	0.5	0.33
Mean	0.5 (0.01)	0.25 (0.0)	0.5 (0.0)	0.33 (0.0)

Table D.108. LSTM (A3) trained on the matrix profile $\mathbf{P}_{25}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.86	0.88	0.86	0.86
2	0.54	0.54	0.54	0.54
3	0.55	0.55	0.55	0.55
Mean	0.65 (0.18)	0.66 (0.19)	0.65 (0.18)	0.65 (0.18)

Table D.109. LSTM (A1) trained on the matrix profile $\mathbf{P}_{[15,25]}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.48	0.5	0.35
2	0.5	0.25	0.5	0.33
3	0.89	0.9	0.89	0.89
Mean	0.63 (0.23)	0.54 (0.33)	0.63 (0.23)	0.52 (0.32)

Table D.110. LSTM (A2) trained on the matrix profile $\mathbf{P}_{[15,25]}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.25	0.5	0.33
2	0.5	0.25	0.5	0.33
3	0.49	0.25	0.5	0.33
Mean	0.5 (0.01)	0.25 (0.0)	0.5 (0.0)	0.33 (0.0)

Table D.111. LSTM (A3) trained on the matrix profile $\mathbf{P}_{[15,25]}$ created from the simulated dataset with torque as the input feature.

Gated Recurrent Unit

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.26	0.5	0.34
2	0.52	0.26	0.5	0.34
3	0.52	0.51	0.5	0.41
Mean	0.52 (0.0)	0.34 (0.14)	0.5 (0.0)	0.36 (0.04)

Table D.112. GRU (A1) trained on the matrix profile $\mathbf{P}_{15}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.26	0.5	0.34
2	0.52	0.26	0.5	0.34
3	0.7	0.82	0.68	0.65
Mean	0.58 (0.1)	0.45 (0.32)	0.56 (0.1)	0.44 (0.18)

Table D.113. GRU (A2) trained on the matrix profile $\mathbf{P}_{15}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.34	0.41	0.33
2	0.48	0.24	0.5	0.32
3	0.7	0.74	0.69	0.67
Mean	0.52 (0.16)	0.44 (0.26)	0.53 (0.14)	0.44 (0.2)

Table D.114. GRU (A3) trained on the matrix profile $\mathbf{P}_{15}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.52	0.26	0.5	0.34
2	0.52	0.26	0.5	0.34
3	0.48	0.25	0.46	0.32
Mean	0.51 (0.02)	0.26 (0.01)	0.49 (0.02)	0.33 (0.01)

Table D.115. GRU (A1) trained on the matrix profile $\mathbf{P}_{25}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.21	0.41	0.28
2	0.52	0.26	0.5	0.34
3	0.48	0.24	0.5	0.32
Mean	0.46 (0.07)	0.24 (0.03)	0.47 (0.05)	0.31 (0.03)

Table D.116. GRU (A2) trained on the matrix profile $\mathbf{P}_{25}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.21	0.41	0.28
2	0.52	0.75	0.54	0.41
3	0.48	0.24	0.5	0.32
Mean	0.46 (0.07)	0.4 (0.3)	0.48 (0.07)	0.34 (0.07)

Table D.117. GRU (A3) trained on the matrix profile $\mathbf{P}_{25}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.48	0.25	0.46	0.32
2	0.48	0.24	0.5	0.32
3	0.48	0.49	0.5	0.38
Mean	0.48 (0.0)	0.33 (0.14)	0.49 (0.02)	0.34 (0.03)

Table D.118. GRU (A1) trained on the matrix profile $\mathbf{P}_{[15,25]}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.57	0.6	0.58	0.54
2	0.48	0.25	0.46	0.32
3	0.43	0.23	0.45	0.3
Mean	0.49 (0.07)	0.36 (0.21)	0.5 (0.07)	0.39 (0.13)

Table D.119. GRU (A2) trained on the matrix profile $\mathbf{P}_{[15,25]}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.39	0.21	0.41	0.28
2	0.57	0.76	0.58	0.49
3	0.52	0.26	0.5	0.34
Mean	0.49 (0.09)	0.41 (0.3)	0.5 (0.09)	0.37 (0.11)

Table D.120. GRU (A3) trained on the matrix profile $\mathbf{P}_{[15,25]}$ created from the original dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.65	0.68	0.65	0.64
2	0.81	0.81	0.81	0.81
3	0.98	0.98	0.98	0.98
Mean	0.81 (0.17)	0.82 (0.15)	0.81 (0.17)	0.81 (0.17)

Table D.121. GRU (A1) trained on the matrix profile $\mathbf{P}_{15}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1*	0.5	0.25	0.5	0.33
2	0.92	0.93	0.92	0.92
3*	0.49	0.25	0.5	0.33
Mean	0.64 (0.25)	0.48 (0.39)	0.64 (0.24)	0.53 (0.34)

Table D.122. GRU (A2) trained on the matrix profile $\mathbf{P}_{15}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1*	0.5	0.25	0.5	0.33
2*	0.5	0.25	0.5	0.33
3	0.92	0.93	0.92	0.92
Mean	0.64 (0.24)	0.48 (0.39)	0.64 (0.24)	0.53 (0.34)

Table D.123. GRU (A3) trained on the matrix profile $\mathbf{P}_{15}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1*	0.5	0.25	0.5	0.33
2	0.8	0.81	0.8	0.8
3	0.68	0.71	0.68	0.67
Mean	0.66 (0.15)	0.59 (0.3)	0.66 (0.15)	0.6 (0.24)

Table D.124. GRU (A1) trained on the matrix profile $\mathbf{P}_{25}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.5	0.25	0.5	0.33
2	1.0	1.0	1.0	1.0
3	0.8	0.8	0.8	0.8
Mean	0.77 (0.25)	0.68 (0.39)	0.77 (0.25)	0.71 (0.34)

Table D.125. GRU (A2) trained on the matrix profile $\mathbf{P}_{25}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.99	0.99	0.99	0.99
2*	0.5	0.25	0.5	0.33
3*	0.49	0.25	0.5	0.33
Mean	0.66 (0.29)	0.5 (0.43)	0.66 (0.28)	0.55 (0.38)

Table D.126. GRU (A3) trained on the matrix profile $\mathbf{P}_{25}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.98	0.98	0.98	0.98
2	0.67	0.68	0.67	0.67
3	0.89	0.9	0.89	0.89
Mean	0.85 (0.16)	0.85 (0.16)	0.85 (0.16)	0.85 (0.16)

Table D.127. GRU (A1) trained on the matrix profile $\mathbf{P}_{[15,25]}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1	0.99	0.99	0.99	0.99
2	1.0	1.0	1.0	1.0
3	0.99	0.99	0.99	0.99
Mean	0.99 (0.01)	0.99 (0.01)	0.99 (0.01)	0.99 (0.01)

Table D.128. GRU (A2) trained on the matrix profile $\mathbf{P}_{[15,25]}$ created from the simulated dataset with torque as the input feature.

Run nr.	Accuracy	Precision	Recall	F1-score
1*	0.5	0.25	0.5	0.33
2	0.99	0.99	0.99	0.99
3*	0.49	0.25	0.5	0.33
Mean	0.66 (0.29)	0.5 (0.43)	0.66 (0.28)	0.55 (0.38)

Table D.129. GRU (A3) trained on the matrix profile $\mathbf{P}_{[15,25]}$ created from the simulated dataset with torque as the input feature.