Intelligent Machine Reliability with General Value Functions

by

Andy Wong

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Statistical Machine Learning

Department of Computing Science

University of Alberta

Abstract

This thesis investigates the use of general value functions for detecting anomalous behavior in machines. Identifying abnormal behavior is critical for ensuring the safety and reliability of any machine or industrial process. When the cause of these anomalies is due to accumulated wear on components over time, maintenance needs to be conducted before failure occurs. The goal of a condition monitoring system is to identify faults that precede machine failure, using only data collected during regular machine operation. Here, we develop a method of using general value functions for the semi-supervised learning problem of machine fault detection.

This method of time series anomaly detection, named General Value Function Outlier Detection (GVFOD), is compared to nine existing methods of novelty detection, including eight multivariate techniques, and one method for time series data. We evaluate these algorithms on a machine failure dataset, collected from a robotic arm previously created at the University of Alberta. The dataset consists overwhelmingly of data collected under normal operation, in addition to five different types of artificially induced failure. It was found that GVFOD outperforms all other algorithms by mean F1-score when sufficient training data is provided, along with fault data for hyperparameter selection. At smaller training sizes, GVFOD performs similarly to multivariate outlier detection algorithms. When default hyperparameters are used, or when selected through expert knowledge without the use of fault data, it was found that GVFOD continues to outperform other algorithms with sufficient

training data, whereas some other algorithms suffer.

Furthermore, a simulation of the robot arm setup was developed using a mechanistic model, and parameter search was used to find unknown material properties and experimental conditions. Using this simulator, gradual failure data was generated and used to compare the performance of GVFOD, UDE (Unexpected Demon Error), and LOF (Local Outlier Factor). These results help us understand why GVFOD is superior to other methods for machine fault detection; the tracking behavior of GVFOD creates a boundary of normality tailored to the tail-end of training data. This allows GVFOD to better identify future normal data, while maintaining its ability to discriminate data arising from faulty operation.

This work demonstrates the challenges of creating effective data-driven machine fault detection systems, and how GVFOD and reinforcement-learning methods are especially suitable for this task. This is a significant improvement upon existing methods of anomaly detection for industrial machine reliability, and contributes to the overall goal of improved system safety and operational efficiency. However, the methods presented in this thesis are generic, and are easily extensible to other fields where anomaly detection in time series data is desired.

Preface

The robot arm simulator in Chapter 4 of this thesis was originally developed by Anthony Maltais and Prof. Michael Lipsett of the Mechanical Engineering Department at the University of Alberta. The model was translated from its original MATLAB code to a Python implementation by myself. The modifications to the model, as well as control, and experimental results were of my own work.

I intend to publish the GVFOD algorithm and its empirical results from Chapter 3 in the future.

To Hylann

For being the best friend anybody could ask for.

Acknowledgements

I would first like to thank my supervisor Professor Osmar Zaïane for his guidance and support in completing this project. Additionally, I would like to thank Professor Johannes Günther for his advice and helpful guidance. Johannes has been a great mentor, and his ideas and knowledge have helped me learn so much through the past two years of my academic career.

Mr. Tomoharu Takeuchi and Mr. Shotaro Miwa and his colleagues at Mitsubishi Electric Co. have graciously provided their time and funding to enable this research. I am so thankful for the consistent feedback and discussions we were able to have. My summer internship in Japan was an experience I will never forget.

I would also like to thank Professor Michael Lipsett and Mr. Anthony Maltais for the their developments in designing and building the robot arm apparatus at the U of A. Without it, I would not have any data to conduct this research.

I am so thankful for Mr. Mohammad Riazi, whose own thesis work paved the way for my research. His help was tremendous in getting me up to speed on this research collaboration.

Contents

1 Introduction **1**
- 1.1 Introduction to Fault Detection . . . 1
- 1.2 State of the Art and Limitations . . . 4
  - 1.2.1 Techniques for Fault Detection . . . 4
  - 1.2.2 Machine Learning for Fault Detection . . . 5
- 1.3 Thesis Statement and Contributions . . . 6
- 1.4 Thesis Layout and Structure . . . 7

2 Background Material **9**
- 2.1 Machine Failure . . . 9
  - 2.1.1 Mechanical Failure . . . 9
  - 2.1.2 Environmental Causes . . . 12
  - 2.1.3 Electrical Failure . . . 13
- 2.2 Outlier Detection . . . 14
  - 2.2.1 Multivariate Outlier Detection . . . 15
  - 2.2.2 Outlier Detection for Temporal Data . . . 25
  - 2.2.3 Markov Chain for Outlier Detection . . . 27
- 2.3 Reinforcement Learning . . . 28
  - 2.3.1 Markov Decision/Reward Processes . . . 29
  - 2.3.2 Temporal Difference Learning . . . 30
  - 2.3.3 Function Approximation . . . 31
  - 2.3.4 General Value Functions . . . 33
- 2.4 Testbench Construction . . . 35
  - 2.4.1 Hardware . . . 35
  - 2.4.2 Software . . . 36
  - 2.4.3 Empirical Fault Data . . . 36
  - 2.4.4 Dynamical System Modelling . . . 37

3 General Value Function Outlier Detection - GVFOD **40**
- 3.1 GVFOD - General Value Function Outlier Detection . . . 40
- 3.2 Experiment . . . 44
- 3.3 Results and Discussion . . . 49
- 3.4 Conclusion . . . 58

4 Fault Detection for Simulated Non-Stationary Machine Behaviour **60**
- 4.1 Robot Arm System Identification . . . 60
  - 4.1.1 Control . . . 64
- 4.2 Experiment . . . 66
- 4.3 Results and Discussion . . . 71
- 4.4 Advantages and Limitations of Simulation . . . 74
- 4.5 Conclusion . . . 74

5 Conclusion **76**

References **78**

Appendix A Extended Results for GVFOD **84**

List of Tables

2.1 Robot arm fault dataset 37

3.1 Algorithms and parameters used 48

4.1 Optimal and empirical parameters for robot arm simulator . . 63
4.2 PID control parameters 65
4.3 State vector and initial values of dynamical system 66

List of Figures

2.1 Typical stress-strain curve in a metal 10
2.2 Tile Coding 32
2.3 Robot arm hardware 36
2.4 Robot arm component names 37
2.5 The dynamical system model of the robot arm given in Lipsett *et al.* [25]. 38

3.1 Proportion of explained variance - PCA 44
3.2 Data for hyperparameter search and algorithm evaluation . . . 46
3.3 Normal data splitting for hyperparameter optimization 47
3.4 Data splitting to evaluate training size requirements 50
3.5 Data splitting to evaluate training size requirements with delay 50
3.6 Performance on hyperparameter tuning data 51
3.7 Performance on validation data 52
3.8 Performance on validation data with delay 53
3.9 Performance on hyperparameter tuning data, using default parameters 56
3.10 Performance on validation data, using default parameters . . . 57
3.11 Performance on validation data with delay, using default parameters 57

4.1 Default performance on robot-arm simulator 61
4.2 Default performance on robot-arm simulator 64
4.3 Trapezoidal velocity profile 66
4.4 Simulated controlled robot arm data 67
4.5 Changing the base belt tension ($F_{t,base}$) in the PID controlled robot arm simulator. 68
4.6 Changing the slope parameter ($slope_1$) in the PID controlled robot arm simulator. 69
4.7 Changing the arm pulley viscous friction (f_{2a}) in the PID controlled robot arm simulator. 69
4.8 Our model of non-stationary machine operation, fault, and failure. 70
4.9 Simulated controlled robot arm data with gradual degradation and fault 71
4.10 Outlier scores of LOF and GVFOD, as well as UDE on gradual failure data 73

A.1 Algorithm comparison on *hyperparameter tuning* data with optimized parameters. No delay. 85
A.2 Algorithm comparison on *validation* data with optimized parameters. No delay. 86
A.3 Algorithm comparison on *validation* data with optimized parameters. 2 hour delay. 87

A.4 Algorithm comparison on *hyperparameter tuning* data with default parameters. No delay. 88
A.5 Algorithm comparison on *validation* data with default parameters. No delay. 89
A.6 Algorithm comparison on *validation* data with default parameters. 2 hour delay. 90

Glossary

ABOD Angle Based Outlier Detection [algorithm]

ARIMA Auto-Regressive Integrated Moving-Average

CBM Condition-Based Maintenance

CM Condition Monitoring [data]

CR Contamination Ratio

GVF General Value Function

GVFOD General Value Function Outlier Detection

HBOS Histogram Based Outlier Score [algorithm]

HMM Hidden Markov Model

ICA Independent Component Analysis

KNN k^{th} Nearest Neighbor [algorithm]

LOF Local Outlier Factor [algorithm]

MC Markov Chain [algorithm]

MCD Minimum Covariance Determinant [algorithm]

MDP Markov Decision Process

MRP Markov Reward Process

MSE Mean Squared Error

OCSVM One Class Support Vector Machine [algorithm]

OD Outlier Detection

PCA Principal Component Analysis [algorithm or decomposition method]

PID Proportional-Integral-Derivative [control]

POF Probability of Failure

POMDP Partially Observable Markov Decision Process

RL Reinforcement Learning

RUL Remaining Useful Life

TD Temporal Difference

UDE Unexpected Demon Error

UTS Ultimate Tensile Strength. Also known as Tensile Strength (TS)

Nomenclature

Dynamical Modelling

m, i, a	subscript notation for motor, idler, and arm [pulleys]
$1, 2, 3$	subscript notation for belt section 1, 2, and 3. See Figure 2.4
I	mass moment of inertia
EA	Young's modulus $\times$ cross-sectional area
l	length
r	radius
k	spring constant
d	damping factor
F_t	tensile force
θ, $\dot{\theta}$, $\ddot{\theta}$	angular position, velocity, and acceleration
τ	torque
f_{1m}, f_{1i}, f_{1a}	dynamic friction parameter
f_{2m}, f_{2i}, f_{2a}	viscous friction parameter
$slope_1, slope_2$	slope parameter
K_p, T_{int}, T_{dt}	PID gain, integration time, and derivative time

General

$\boldsymbol{x}$	a vector representing a sample, record, or observation in $\mathbb{R}^{Tk}$
$\boldsymbol{X}$	a sample matrix consisting of samples as its rows
$\boldsymbol{\mu}$	mean vector
$\boldsymbol{\Sigma}$	variance-covariance matrix
$score$	outlier score
T	number of ticks per period. Equivalent to sampling rate (Hz) $\times$ period (s)

Reinforcement Learning

$\boldsymbol{s}, \boldsymbol{s'}, S$	state
A	action
R	reward
C	cumulant
γ	discount-rate
$v(s)$	state-value of state s
$v_C(s)$	general value of state s for cumulant C
$\hat{v}$	estimated value
$\boldsymbol{x}$	observation vector
$\boldsymbol{\phi}$	feature vector
$\boldsymbol{w}$	weight vector
$\boldsymbol{z}$	trace vector
λ	trace-decay parameter
α	step-size
δ	temporal-difference error

Chapter 1

Introduction

1.1 Introduction to Fault Detection

One of the greatest impediments to the automated production of goods is equipment malfunction. In the 200 years since the first industrial revolution, machines have taken over what was previously done by hand, at a scale that would have been impossible with only human labour. With these gargantuan advances, the cost of downtime due to equipment failure has grown proportionally, and can now reach upwards of millions of dollars per hour [34][5]. Ensuring reliability of industrial processes starts with good process design, but given sufficient time, even the best-engineered machines will deteriorate. Maintenance is necessary to keep the equipment in a good state of repair.

Two classic maintenance techniques are breakdown maintenance and scheduled maintenance [21]. Breakdown maintenance occurs when machines are fixed after they have broken. Scheduled maintenance occurs periodically, with parts preventatively replaced regardless of their state of wear. Breakdown maintenance, often employed for consumer products and systems with low cost of failure, can lead to extended downtimes and increased costs since the operator needs to react to an unexpected event. There is a loss of revenue from delayed production, and there may be damage caused to nearby machinery and injury to personnel. Scheduled maintenance, often seen in higher cost-of-failure systems, can lead to an excess of servicing costs, since parts are replaced before it is necessary, and the process is interrupted to facilitate this service. It also requires up-front engineering costs, where the life of a compo-

nent needs to be estimated using expert knowledge. Scheduled maintenance does not consider the variance in individual component quality, nor the intensity and environment of its use, instead just considering a worst-case scenario and compounding the risk into a general safety factor.

In the space between the extremes of breakdown and scheduled maintenance, is condition-based maintenance (CBM). CBM utilizes information about the current condition of the machinery to determine the necessary maintenance activities. The sensors used for CBM can be targeted for specific modes of failure. For example, a vibration sensor can be installed on a bearing, and a strain gauge can be installed on a beam to measure the deflection under load. However, we want to investigate the utility of sensors not designed for condition monitoring towards the goal of CBM. These sensors may exist for a variety of reasons, including feedback control, machine calibration, or if the sensory output was the intended output of the process. If the existing sensors on a machine can be used to monitor its health, CBM can be retrofitted to old machines. Likewise, these same algorithms could be used to achieve cost efficient CBM on new machines, with no need for additional sensors.

Jardine *et al.* [21] describes the CBM methodology in three major parts:

1. Data Acquisition, to collect relevant data for system health
2. Data Processing, to handle and transform the data
3. Maintenance Decision-Making, to analyze the data to recommend efficient maintenance

This holistic view of machine maintenance encompasses more than just fault detection. The first step, data acquisition, includes collecting condition monitoring (CM) data as well as event data (times and details of maintenance activities, operational events, etc.). With modern technology, condition monitoring data can be stored and processed automatically. Event data still needs to be collected with some level of human interaction, but is crucial for feedback for the condition monitoring system; in a machine learning context, event data can act as labels, used for supervised/semi-supervised learning. The collected

data is stored, and made available for data processing. Historically, event data has been used extensively in the field of reliability analysis. The second step, data processing, transforms raw condition monitoring data into a form more useful for maintenance decision making. The possibilities include no processing, fourier transforms into the frequency domain, some combination of time-frequency analysis using wavelets, among others. A time series analysis technique like Auto-Regressive Integrated Moving Average (ARIMA) can be used, to find a set of coefficients matching the ARIMA model. Statistical methods like principal component analysis (PCA) and independent component analysis (ICA) can be used to reduce the dimensionality of condition monitoring data. After these transformations are completed, the processed data can be used for maintenance decision making. This consists of two major components:

- Fault Diagnostics, including
  - Fault Detection
  - Fault Classification
- Fault Prognostics

Fault detection, the topic of this thesis, considers the question *"Is there something wrong with the machine?"*, as opposed to fault classification, which considers *"What is the problem with the machine?"*. Fault prognostics is a more difficult problem for CBM, asking questions like "What is the remaining useful life (RUL) of the machine?" and "What is the probability of failure (POF) on the machine within the next week/month/year etc.?". The data requirements of each task in maintenance decision making differ substantially. Fault detection requires only normal-operation data to learn from. Fault classification requires either CM data under different faulty situations, or expert knowledge, in order to determine the type of failure. Meanwhile, fault prognostics requires event data, in combination with expert knowledge in order to accurately find the RUL or POF.

In this thesis, we define machine fault and machine failure as follows. Machine failure has occurred when the machine has worn down until it can no longer perform its task to a satisfactory level. Machine fault is an event that indicates machine failure is incipient. Two ideas are implied in this definition of machine failure. Firstly, machine degradation is gradual. A sudden failure, such as electronic damage from a power surge, or physical damage from an external impact, does not have any reasonable warning signs that a CBM program could detect. Secondly, machine failure can occur without a catastrophic stoppage in the process. If the product quality has dropped beyond an acceptable quality, failure has occurred. For example, in steel plate production, if steel plate thicknesses are beyond tolerable limits, failure has occurred, even though the process has not stopped.

1.2 State of the Art and Limitations

This thesis investigates the use of condition monitoring (CM) data for fault detection. An overview of current methods and their limitations is described here.

1.2.1 Techniques for Fault Detection

Engineered fault detection methods are the most direct methods of fault detection. In these applications, a sensor is specifically installed for the purpose of detecting faulty behaviour arising from a single failure mode. For example, for a component where there is a risk of overheating, a thermocouple can be installed and monitored. For a shaft or rotating assembly that is known to become unbalanced over time, a vibration sensor can be installed to ensure that the imbalance remains tolerable. Engineered fault detection methods are highly explainable, which consequently makes it relatively simple to set limits on acceptable behaviour using expert knowledge. The utility of engineered fault detection goes beyond maintenance, and can be used to guarantee the safety of a process against specific modes of failure. If safe operating limits are exceeded, alarms can be raised, and if necessary, the machine can be shut

down. Certain systems have automatic shut-down procedures, bringing the entire process to a safe halt without human intervention. However, engineered fault detection is not cheap. By definition, these sensors can be removed from the machine and it would still be operable. Likewise, there is a development cost for these targeted maintenance and condition monitoring systems, and it may be inconvenient to retrofit them into existing machines. Finally, a targeted fault detection method is precise - it likely will not be able to detect a fault that was not anticipated in the design phase.

Data-driven fault detection methods are an approach to fault detection through indirect means. This usually means that the machine is more affordable. Two possibilities for data-driven fault detection exist - model-based, and model-free. A model-based fault detection system uses a physics-based model of the machine in order to simulate its behaviour in real-time. Condition monitoring data, collected on the real machine, is compared with simulator values. Discrepancies between the model and empirical data can be explained through sensor error, random noise, or most critically - the dynamics of the machine have changed. This is a potential indicator of failure. Although not as explicit as engineered fault detection, there is some explainability, and the source of the fault could be interpreted depending on which sensors are behaving anomalously. Developing a simulator is difficult, and requires, at the minimum, extensive expert knowledge. For a simple machine, we highlight specific challenges for developing a simulator in Chapter 4. For a complex system, it may be completely unfeasible to develop a simulator. This can be caused by having too many unidentifiable parameters in the model. Or, it can be too computationally expensive to evaluate the model in real-time. Model-free methods, (used synonymously in this thesis with machine learning methods), are an alternative to model-based methods.

1.2.2 Machine Learning for Fault Detection

Using machine learning and outlier detection methods for fault detection in machines was previously studied in Riazi *et al.* [35] [36]. Not only did Riazi *et al.* provide a comprehensive review of this subject, they also created an

empirical dataset suitable for evaluating fault detection algorithms. Outlier detection is a semi-supervised learning technique. Given a dataset consisting of normal exemplars, the algorithm draws a boundary of normality. When presented with test data, it determines whether that new data is within or beyond this boundary of normality. An overview of the algorithms tested in this thesis is available in 2.2.1. In general, Riazi found that all outlier detection algorithms could differentiate between faulty and normal operation [36], with no statistically significant difference in performance between them. This thesis complements and builds upon the findings in Riazi *et al*.

There are two limitations to the previous study that deserve further consideration. Firstly, Riazi [36] considers each sample from the normal operating condition to be independent and identically distributed. However, for real machinery, there can be drift between a number of normal operating conditions, and training on past normal data may not generalize well to future normal operating data. Secondly, all the outlier detection algorithms tested were general multivariate methods, and not explicitly designed for time series. For this study, each data sample consists of time series of equal length from multiple sensors. If the elements in each sample were reordered, multivariate outlier detection algorithms would perform identically. Intuitively, there is more information to be exploited, which we explore in this study.

1.3 Thesis Statement and Contributions

This thesis seeks to answer the following question

Can we use general value functions and temporal difference learning with native data to detect faults and assess machine health in machines?

Native data is defined here as condition monitoring data collected from sensors that were not installed specifically for engineered fault detection. In the process of answering this research question, we have made the following contributions.

We developed the General Value Function Outlier Detection (GVFOD) algorithm, a general value function (GVF) based outlier detection technique

specifically designed for predicting machine failure. The algorithm is based upon surprise and unexpected demon error (UDE) [48]. UDE was shown to be successful in detecting local additive outliers in machine operating data [15]. By adapting that algorithm to use a fixed model, it is better suited to detect machine faults, which present as innovative outliers. This algorithm is introduced in Chapter 3.

We compare this new algorithm with existing outlier detection algorithms for the purposes of machine fault detection. Instead of running experiments with shuffled data and k-folds cross-validation, we have elected to evaluate algorithms based upon how they would be used in the real world - by training on historical data, and testing on future data. This accomplishes two goals - it enables us to use reinforcement learning algorithms like GVFOD, which requires temporal continuity (a "stream of experience"), and it lets us evaluate outlier detection methods in a setting that can be more challenging, due to non-stationarity in the machine's behaviour.

Previous contributors to this research agreement have created a robot-arm testbench, along with a preliminary dynamical model describing its operation. We completed the development of this model for the purposes of generating simulated data suitable for evaluating fault detection algorithms. The model consists of twenty parameters, all of which have physical meaning, which can modified in order to simulate real machine failure. The process of developing this model is of potential utility to researchers and engineers who may have an idea of how to model their machinery, but without the ability to test everything empirically. Instead, unknown parameters are learned from data.

1.4 Thesis Layout and Structure

The following chapters of this thesis are laid out in the following manner.

Chapter 2 goes over the necessary information required to understand this thesis. Included within is an explanation of the problem of machine failure, the multivariate and time series outlier detection algorithms which we use to predict failure, an introduction to reinforcement learning and general value

functions, and an overview of testbench development previously by others, on which our own developments build upon.

In Chapter 3, we go over the GVFOD algorithm, and evaluate its performance compared to multivariate outlier detection algorithms on the dataset for machine fault detection. This dataset consists of time series data collected from a robot arm apparatus, with labels for normal and faulty operating conditions. Semi-supervised experiments are done with this dataset to investigate the training data needs of different algorithms, and how learning from the past generalizes into accurate predictions of failure in the future.

Lastly, in Chapter 4, we develop a simulator to generate new failure data. We modify an existing simulator, tuning parameters with a hyperparameter search algorithm, and add a control method to ensure consistent generation of data. Using this, we are able to create gradual failure data, and we can create an intuition for the inner workings of the GVFOD algorithm. This includes a comparative evaluation of a representative multivariate outlier detection algorithm, GVFOD, and unexpected demon error (UDE).

The last chapter concludes this thesis, where we will briefly discuss and summarize these results, and present some potential areas of further research.

Chapter 2

Background Material

2.1 Machine Failure

A brief overview of the possible causes of failure in industrial machinery is outlined below. They are grouped broadly into mechanical causes, environmental causes, and electrical causes. The goal of CBM is to predict failure caused by accumulated wear on a component of a machine. In an outlier detection setting, this accumulated wear would present as innovative outliers (or level shift) - where the deviation from normal behaviour grows over time. This is in contrast to additive outliers - more indicative of an unpredictable[1] failure.

2.1.1 Mechanical Failure

A machine has failed mechanically when some component has failed due to the application of physical force.

Two terms commonly used to describe a material's response to the application of a force are stress and strain. Stress is the force applied per unit area, whereas strain is a unitless measure of the relative change in dimension of the material. The strain of a component under load can be measured using a strain gauge, and with knowledge of the material properties and its shape, the stress can be calculated using a stress-strain curve. A typical stress-strain curve is shown in Figure 2.1

From a materials science and engineering point of view [6], mechanical fail-

[1] unpredictable using CBM methodology, but most likely predictable using physics and engineering knowledge

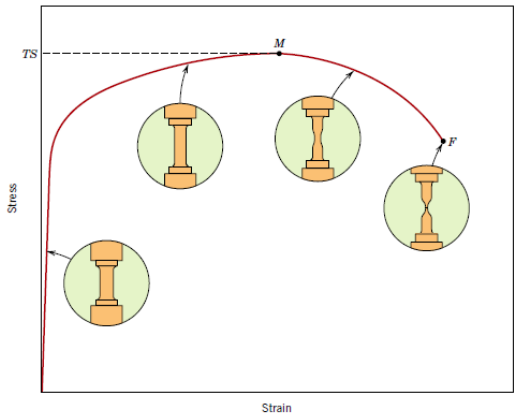


Figure 2.1: A typical engineering stress-strain curve for a metal. The ultimate tensile strength (TS/UTS) is shown. The transition from elastic deformation (linear) to plastic deformation (non-linear) occurs at the elbow on the left hand side of the curve. Image source: Callister and Rethwisch [6].

ure occurs when a fracture has developed within a component. Fracture can be classified as either brittle or ductile based upon visual inspection of the broken component. Brittle fracture occurs when the component breaks with no plastic (permanent) deformation. Ductile fracture occurs when there is plastic deformation at the break - and the material's shape had changed permanently prior to fracturing. When a component is improperly engineered, or loaded beyond its design envelope, the internal stress will exceed the ultimate tensile strength (UTS) and fracture will occur. The effect of this can be seen by excessive deflection, buckling, and in most materials, some sort of plastic deformation. In general, this type of material failure is not consistent with the goals of CBM, since this failure mode is not attributed to component wear over a period of use. Rather, it happens the first time the load on the machine exceeds what the material allows. In order to prevent this kind of failure, proper engineering and testing is required.

Three kinds of material failure that are due to accumulated wear are fatigue, creep, and mechanical wear.

Fatigue is material failure that occurs when a crack propagates through a component due to repeated application of stresses less than the UTS. Examples of components that could fail due to fatigue include shafts and bearings, vessels

where the internal pressure varies, and parts that vibrate. Any component that cycles between loading and unloading can encounter fatigue failure. Fatigue failures are responsible for 90% of failures in metals, and affect plastics and composites too. Fatigue failure initiates from an imperfection in the part; where there is a stress concentration - an area where, due to geometry, local stress is greater than the bulk material stress. A crack initiates; then with each loading cycle, the crack grows a small amount, separating the material a little more. Over time, the cross-sectional area of the component decreases, and the crack propagation accelerates, until the part fails catastrophically. A possible symptom of fatigue failure could be increased deflection to a load, as the growing crack reduces the effective stiffness of the component. This can be very difficult to detect. In materials with low fracture toughness (like ceramics), fatigue failure may be unpredictable, since a small defect will quickly propagate through almost the entire cross-section.

Creep is a mode of failure where a constant applied stress on a component under a high temperature causes it to stretch out over time. Creep affects metals, polymers, and ceramics alike. Note that this is different from melting: in metals, creep becomes a problem at temperatures around $0.4T_m$, where T_m is the absolute melting temperature of the metal. For polymers, creep can be very significant even at room temperature and with low stresses. Ceramics, in general, require a higher temperature than polymers and metals before they exhibit creep. In a machine, creep will cause the dimensions of a component to change, potentially causing problems with operation. With sufficient time, the geometry will have changed sufficiently leading to an inevitable fracture.

Mechanical wear is the process where a component wears by having two contacting surfaces slide relative to one another. The friction between the two surfaces causes material to slough off. Some forms of mechanical wear are a part of design - for example, brake pads on the brake rotor of a motor vehicle. Sometimes mechanical wear is unintended; examples include improperly lubricated surfaces, or a design and manufacturing error.

Of the modes of failure described above, the ones that CBM aims to prevent are the gradual ones - fatigue, creep, and mechanical wear. These gradual

failures all have a notion of accumulated degradation. On the other hand, mechanical failure that is not dependent on the duration and severity of machine operation is much more difficult to predict. These situations, where a one-time load exceeds what the component can handle (due to improper design, manufacturing, misuse, or accidents), are not a problem that CBM is intended to solve.

2.1.2 Environmental Causes

The environment can cause gradual degradation to materials.

Corrosion and oxidation are processes where metals are [unintentionally] chemically attacked. The canonical example of corrosion is the oxidation of iron and its alloys. In the presence of water and oxygen, untreated iron will turn from its metallic form (Fe) to rust ($Fe(OH)_3$). Methods which are used to mitigate corrosion in steel exposed to the environment include physical barriers like paint and polymer coatings, chemical treatment like galvanization, passivity by alloying with other elements (stainless steel), and active methods like cathodic protection.

Metals that are typically not susceptible to corrosion can become so when they are stressed. For example, aluminum in normal environments forms a passivizing layer of aluminum oxide which prevents metal underneath from wearing. However, when stressed, cracks can form and grow. This combination of static or dynamic loading and a corrosive environment causes stress-corrosion cracking (SCC), which can further evolve into fatigue cracking.

Ceramics, unlike metals, rarely degrade in normal environments. Under extreme environments, they may simply dissolve, but do not react chemically like metals. For these reasons, ceramics are used commonly where exposure to harsh chemicals or high temperatures is needed.

The degradation of polymers is usually physical, or a combination or physical and chemical processes. Many polymers will swell in the presence of a susceptible solvent. The solvent molecules diffuse into the polymer structure and change its physical properties. Depending on the solvent, some polymers will simply dissolve. The process by which the chemical bonds in a polymer

breaks is called scission. This can happen with exposure to chemicals, thermally, or with radiation. The symptoms of this mode of failure are similar to those of creep.

Environmental causes of failure are typically gradual. Since they often cannot be wholly prevented, these modes of failure are all potential candidates for CBM, to ensure appropriate remediation can take place before a machine fails.

2.1.3 Electrical Failure

Compared to mechanical systems, electronics are usually more intricate and complex. In industrial settings, the physical size of electrical systems is typically far smaller than its mechanical counterpart. Failures of these electronics happen on the micro or nano scale, as opposed to the macro scale of mechanical failures. Electronic failures can sometimes be intermittent, and some faults may not even lead to any change in functionality until failure occurs. All of these factors couple with the continued miniaturization of electronics, so the tolerance for defects continues to tighten. Prognostics and Health Management (PHM) for electronics is therefore significantly more difficult than its mechanical counterpart [47] [32].

The failure mechanisms of electronics are not consistently defined in literature. Pecht [30] provides an overview of PHM for electronics, and identifies the following failure mechanisms which are common among wear-out failures.

- Fatigue (from thermal and/or mechanical stress)
- Corrosion
- Electromigration
- Conductive filament formation
- Stress driven diffusion voiding
- Time-dependent dielectric breakdown

The failure mechanisms arise from a variety of loads. Whereas mechanical failure is always caused by mechanical stress, electrical failure can arise from thermal loads, voltage gradients and current density, and mechanical stress; some of which also interact with environmental effects like humidity, moisture, and mean temperature.

Just as in mechanical failure, electronics failure from over-stress events (single loads that exceed the component's strength) cannot be easily predicted using CBM and PHM. Of concern is that wear-out failures of electronics can progress steadily, but may be asymptomatic. In order to mitigate this risk, engineered solutions of PHM are necessary. Thermocouples can be used to monitor the temperature of components, and strain gauges and accelerometers can measure the mechanical loads. Outside of direct measurement using diagnostic sensors, canary devices can be installed alongside the electronic components. These are specifically designed components that experience the same loads as the functional component, but wear at an accelerated rate. The intention is for the canary to fail before the functional part, to ensure service and replacement of the main component before failure.

2.2 Outlier Detection

This section describes the problem of outlier detection in generic data (multivariate outlier detection), and algorithms to do so. Furthermore, we introduce outlier detection for time series, and the use of Markov models for doing so. We hope to apply outlier detection for detecting faulty behaviour in machines. This is an example of semi-supervised learning, where an algorithm learns from a training set consisting of only normal data, and future observations are classified as normal or abnormal accordingly [2].

[2]In this thesis, we use the terms normal and inlier interchangeably; in the same way, the terms abnormality, outlier, novelty, anomaly, and fault are interchangeable

2.2.1 Multivariate Outlier Detection

An anomaly is an "observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" [19]. Anomaly detection is, in general, a difficult problem to define, since the ground truth is often arbitrarily decided. The terminology used is also confusing and poorly differentiated, especially across different fields of study. Nonetheless, Hodge and Austin [20] divides anomaly detection into three subproblems: supervised anomaly detection, unsupervised anomaly detection, and semi-supervised anomaly detection.

Supervised anomaly detection is imbalanced classification framed as an anomaly detection problem [7]. To conduct supervised anomaly detection, a training dataset is required which consists of labelled normal and abnormal data. Based upon these labelled data points, a classification algorithm will devise a procedure to evaluate future observations, and whether they lie within the area of normality or the area(s) of abnormality. This classification is imbalanced, since in the majority of cases, the available normal data greatly outnumbers the abnormal data. In the context of machine failure detection, supervised anomaly detection is generally infeasible, due to the lack of abnormal data. Only when data originating from a machine operating in both normal and faulty modes is available, can supervised anomaly detection be done.

Unsupervised anomaly detection is the process of identifying observations in a dataset that contains mixed normal and abnormal data. The difference between this and *supervised anomaly detection* is the availability of labels - in the unsupervised setting, no labels are available to learn from. Instead, outliers are the observations that appear to arise from a different distribution that the remainder of the data. This is often used as data preprocessing, as some statistical and machine learning methods are sensitive to the presence of outliers: where a proportionally small number of outliers can disproportionately affect the results of the statistical or machine learning procedure. This sensitivity can negatively affecting the accuracy of the results,

and preprocessing with outlier detection can be used to increase the robustness of the method.

Semi-supervised anomaly detection is the process by which an algorithm learns a boundary of normality on a dataset of normal observations. If a future observation is beyond this boundary, it is classified as abnormal (an outlier, or a positive observation), and as normal (an inlier, a negative observation) otherwise. More often than not, one does not want to draw a boundary surrounding all of the normal data, as there may be some contamination due to noise or measurement error. Instead, the boundary should contain a large majority of the data. For all outlier detection algorithms, a contamination ratio (CR) is designated, which designates the probability that a future inlier will be incorrectly classified as an outlier; and equivalently, the proportion of training data that lies beyond the outlier threshold. The CR can also be thought of as the false positive rate on the training data. In practice, setting the CR is difficult: a low contamination ratio will create an overly large area within the outlier threshold, and future abnormal observations will be misclassified. A high CR will lead to a high false positive rate, and true normal observations will be designated as outliers. Applications of outlier detection include detecting criminal activity, rare events, or in general, finding special cases that may be worth investigating [4]. In this thesis, we use semi-supervised anomaly detection for machine failure detection.

In the available literature, these three sub-problems may be difficult to differentiate based only on the terminology [7]. An author may refer to any one of them as anomaly detection, outlier detection, novelty detection, fault detection, rare event detection, etc. Further compounding the confusion is the similarity between unsupervised anomaly detection and semi-supervised anomaly detection. A semi-supervised anomaly detection, trained on normal data, and tested on the same data, is equivalent to unsupervised anomaly detection. For the purposes of this thesis, I will use *outlier detection* or *OD* to represent semi-supervised anomaly detection. Anomalous observations will be synonymous with anomalies, outliers, abnormalities, and faults.

The most succinct delineation of the different classes of semi-supervised

anomaly detection algorithms is presented in Pimentel *et al*. [31].

1. Probabilistic - model a probability distribution over the normal class of data

2. Distance-based - Measure the proximity to inliers

3. Reconstructed-based - Measure the error when a sample is compressed and uncompressed, compared to reconstruction error optimized for normal data

4. Information-theoretic - measure the information content of a dataset with and without a sample

5. Domain-based - Model the outlier boundary explicitly

These divisions of the outlier detection algorithms is fairly straightforward. For methods that do not fit into the first four groupings, domain-based methods serves as the catch-all class, since the goal of all outlier detection algorithms is to find a boundary of normality. Other reviews of outlier detection [20][8], anomaly detection [9][36], and novelty detection [28][27][31], have different classification of methods; some classify them depending on their originating field of research, like *statistical*, *classification*, and *machine learning*, but this gives little intuition for the mechanisms of the algorithms. Other groupings, like *distance-based* and *density-based*, and *nearest-neighbour-based* methods, are not distinctive enough to warrant individual classes for each one.

There exist open source implementations of many common OD algorithms, an example of this is PyOD [50]. In general, they function as follows: an outlier score is assigned for every training and testing sample. An observation is classified as an outlier when its outlier score is greater than a threshold value; this threshold is a quantile (based on the CR) of the outlier scores in training data. This convention, where outliers have higher scores, is maintained throughout this thesis.

Probabilistic Outlier Detection

Probabilistic outlier detection methods learn a probability density function (density) over normal data. An observation that occurs in an area of low density is more likely to be an outlier. The negative density of an observation (or some monotonic mapping of it) can be used as the outlier score, allowing us to keep the convention that a higher outlier score indicates more outlying behaviour.

Probabilistic methods of estimating a density function can be parametric or non-parametric. Parametric methods assume that the underlying data is sampled from a known distribution, and are defined with a finite set of parameters $\boldsymbol{\theta}$. These parameters are then estimated from normal data, using methods such as maximum likelihood estimation (MLE), Expectation-Maximization (EM), and others. Non-parametric methods make fewer assumptions on the underlying distribution of the data, and the size of the model can expand with additional training data. Note that the distinction between parametric and non-parametric methods is difficult to ascertain - for example, some authors consider histogram based density estimation as non-parametric, when it could be considered a [parametric] mixture of uniformly distributed continuous variables.

The **Minimum Covariance Determinant** (MCD) [37] is a parametric probabilistic method which assumes that the majority of normal data is sampled from a multivariate normal distribution:

$$p(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{d}{2}}\sqrt{\det \boldsymbol{\Sigma}}} \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\intercal}\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$$

Instead of calculating model parameters with the maximum likelihood estimates, MCD finds a more robust estimate of these values $(\hat{\boldsymbol{\mu}}, \hat{\boldsymbol{\Sigma}})$. The MCD estimates of model parameters are made using the h data points that minimize $\det(\hat{\boldsymbol{\Sigma}})$. This estimate is more robust to outliers in the training set, and is equivalent to the maximum likelihood estimate when $h = n$. The algorithm used, FastMCD is described in [38]. The outlier score used is the Mahalanobis

Distance from the centre of the training data:

$$score_{MCD}(\boldsymbol{x}) = \sqrt{(\boldsymbol{x} - \hat{\boldsymbol{\mu}})^{\intercal} \hat{\boldsymbol{\Sigma}}^{-1} (\boldsymbol{x} - \hat{\boldsymbol{\mu}})}$$

Histogram Based Outlier Score (HBOS) [13] is a non-parametric method of learning the distribution from which the training data was sampled from. By assuming the independence of every feature, HBOS estimates the joint density of any observation as a product of the marginal densities of each feature. For each feature, a simple histogram is constructed from the training data. The pre-defined number of bins per dimension, k, along with the maximum and minimum of each feature, is used to calculate the locations of the equal-width bins. By scaling all the bin heights so they have a total area of 1, a valid probability distribution is found. Because HBOS assumes the independence of each feature, computation can occur for each feature concurrently, yielding a fast and simple method for outlier detection. Outlier scores for each observation is calculated as a negative log likelihood:

$$score_{HBOS}(\boldsymbol{x}) = -\sum_{j=1}^{d} \log(hist_j(\boldsymbol{x}_j))$$

Distance-Based Outlier Detection

These are methods reliant on a measure of proximity to other normal observations. These include nearest-neighbour methods, local density measures, and measures of distance between three points using inscribed angles. Observations are more likely to be outliers if they are a large distance away from observations in the training set, and more likely to be inliers otherwise.

k$^{\text{th}}$ Nearest Neighbour (KNN) for outlier detection [33] assigns an outlier score for each observation $\boldsymbol{x}$, as its distance to its k$^{\text{th}}$ nearest neighbour in the training data $\boldsymbol{x}^{(\boldsymbol{k})}$. k is a hyperparameter that needs to be set before learning occurs. An additional hyperparameter is the distance metric used,

which is usually one of the standard L_p norms: L_1, L_2, or L_∞.[3]

$$score_{KNN}(\boldsymbol{x}) = d_p^{(k)}(\boldsymbol{x}) = d_p(\boldsymbol{x}, \boldsymbol{x}^{(k)}) = \left(\sum_{j=1}^{d} (\boldsymbol{x}_j - \boldsymbol{x}_j^{(k)})^p \right)^{1/p}$$

Local Outlier Factor (LOF) [4] is a density-based algorithm that identifies outliers based on neighbourhood density. LOF differs from KNN by finding local outliers in addition to global outliers. Whereas global outliers are observations that appear different from the whole of the training data, local outliers are those that appear different from its closest neighbours. For each sample, its neighbourhood of the k-closest neighbours is defined. An outlier is defined when its own neighbourhood density is much lower than the densities of its neighbours. In LOF, the distance to the k^{th} nearest neighbour is used, identical to $score_{KNN}(\boldsymbol{x}) = d_p^{(k)}(\boldsymbol{x})$ defined above. Using this, a non-symmetric measure of distance between two observations - reachability-distance - is defined as $rd_k(\boldsymbol{a}, \boldsymbol{b}) = \max(d_p^{(k)}(\boldsymbol{b}), d_p(\boldsymbol{a}, \boldsymbol{b}))$. The LOF algorithm works as follows:

1. For every record, find its *Local Reachability Density*:

$$LRD_k(\boldsymbol{x}) = \left(\frac{\sum_{y \in N_k(\boldsymbol{x})} rd_k(\boldsymbol{x}, \boldsymbol{y})}{\mid N_k(\boldsymbol{x}) \mid} \right)^{-1}$$

   Note that $N_k(\boldsymbol{x})$ is the neighbourhood of points within $d_p^{(k)}(\boldsymbol{x})$ of $\boldsymbol{x}$. Its cardinality is equal to k except when there are multiple samples that are equal distance to $\boldsymbol{x}$.

2. Compare the LRD of $\boldsymbol{x}$ to the LRDs of its neighbours in $N_k(\boldsymbol{x})$

$$score_{LOF_k}(\boldsymbol{x}) = \frac{\frac{1}{|N_k(\boldsymbol{x})|} \sum_{\boldsymbol{y} \in N_k(\boldsymbol{x})} LRD_k(\boldsymbol{y})}{LRD_k(\boldsymbol{x})}$$

The LOF of records in the middle of a cluster will be close to 1, and the LOF of global and local outliers will be greater.

Angle Based Outlier Detection (ABOD) [24] attempts to alleviate the problems of distance-based metrics in high dimensional space. This problem

[3]The L_1 norm is the Manhattan distance. The L_2 norm is the Euclidean distance. The L_∞ norm is the Chebyshev distance.

is termed the "Curse of Dimensionality", where in higher dimensions, the distance between points concentrates towards a single value. One method of intuiting this behaviour is by looking at the chi-squared distribution, $\chi^2_d(\alpha)$. The chi-squared distribution describes the Euclidean (L_2) distance of a standard multivariate normally distributed variable from the center. In 2 dimensions, 1% of points is further than $\chi^2_2(0.01) = 9.2$ units away from the center, and the 1% are closer than $\chi^2_2(0.99) = 0.020$, a factor of 500 times. In 100 dimensions, $\chi^2_{100}(0.01) = 135.8$, $\chi^2_{100}(0.99) = 70.065$, a difference of only 1.9 times. Because of this effect, distances become less meaningful in higher dimensions. ABOD addresses this problem by looking at the inscribed angle between a record, and two other samples in the training set. If the angle is small, the observation is more likely to be an outlier.

$$ABOD(\boldsymbol{x}) = -VAR_{\boldsymbol{y},\boldsymbol{z}\in\mathcal{D}}\left(\frac{(\boldsymbol{y}-\boldsymbol{x})^\intercal(\boldsymbol{z}-\boldsymbol{x})}{\|\boldsymbol{y}-\boldsymbol{x}\|^2\|\boldsymbol{z}-\boldsymbol{x}\|^2}\right)$$

In order to better identify local outliers in addition to global outliers, ABOD weighs the scores of closer pairs more than farther-away pairs $\boldsymbol{y}, \boldsymbol{z}$ in the training set $\mathcal{D}$. To increase performance, an alternative algorithm, FastABOD, replaces the entire training set $\mathcal{D}$, with the points in the test record's k-neighbourhood, $N_k(\boldsymbol{x})$. Note that because ABOD and FastABOD still rely on a measure of distance, they do not fully overcome the curse of dimensionality.

$$score_{FastABOD}(\boldsymbol{x}) = -VAR_{\boldsymbol{y},\boldsymbol{z}\in N_k(\boldsymbol{x})}\left(\frac{(\boldsymbol{y}-\boldsymbol{x})^\intercal(\boldsymbol{z}-\boldsymbol{x})}{\|\boldsymbol{y}-\boldsymbol{x}\|^2\|\boldsymbol{z}-\boldsymbol{x}\|^2}\right)$$

Reconstruction-Based Outlier Detection

Oftentimes, a multivariate observation is a high dimensional representation of a set of latent variables in a lower dimensional space. By learning a method to compress, view, or project training data to a lower dimensional space, such that its reconstruction is similar to the original observation, a model of normality can be obtained. An outlier will not be as easily represented using the same reconstruction model, and outlier scores can be assigned based on reconstruction error.

Principal Components Analysis (PCA) is a method of decomposing a random variable into a linear combination of orthogonal vectors (eigenvectors). The eigenvectors that point in the directions of greatest variance can be used to roughly estimate the data. Any deviation that is not captured by the k major eigenvectors is calculated as a weighted sum of squares, and used as the outlier score.

1. Calculate the covariance of the [centred and whitened] data $\boldsymbol{X}$.

$$S = \frac{1}{n}\boldsymbol{X}^\intercal \boldsymbol{X}$$

2. Calculate the eigenvectors $\boldsymbol{e}_i \in \mathbb{R}^d$ and eigenvalues $\lambda_i \in \mathbb{R}$, $i \in \{0, 1, ..., d\}$ of S by some process like SVD. The eigenvectors all have unit length, and the corresponding eigenvalues non-negative and arranged in decreasing order.

$$S = \sum_{i=1}^{d} \lambda_i \boldsymbol{e}_i \boldsymbol{e}_i^\intercal$$

3. For any record, calculate its outlier score by its weighted distance to the plane spanned by the largest n eigenvectors.

$$score_{PCA}(\boldsymbol{x}) = \sum_{i=k+1}^{d} \frac{\boldsymbol{e}_i^T \boldsymbol{x}}{\lambda_i}$$

Information-theoretic Outlier Detection

Information theoretic outlier detection measures the information content in a dataset. Outliers are identified when their presence "alters the information content in the dataset" [31].

One information-theoretic method of identifying outliers is found in Keogh et al. [23]. Data is discretized into strings, and the Kolmogorov complexity is calculated. The Kolmogorov complexity is a measure of the length of instructions needed to produce that record, and more complex records are more likely to be outliers. No information-theoretic outlier detection techniques are evaluated in this thesis, but the ideas of time series discretization and *surprise* are explored in Markov Chain outlier detection and GVFOD respectively.

Domain-Based Outlier Detection

This catch-all class contains all the methods which explicit learn a boundary around the normal data. New exemplars are evaluated in comparison to this boundary.

One-class Support Vector Machines (OCSVM) [39] is a domain-learning algorithm that extends previous work in support-vector classifiers [11] to the case of outlier detection. The original SVM for binary classification begins with the idea of finding a hyperplane that separates the data, where all the data of one class lies on one side of the plane, and vice versa. However, in the case where the data is not linearly separable, slack variables are introduced, penalizing observations that are on the "wrong" side of the plane. Additionally, a feature map can be used to lift the variables $\phi : \mathcal{X} \mapsto \mathcal{F}$ into a higher dimensional space, so that a separating hyperplane can be found. In order to extend this idea, OCSVM aims to find a separating hyperplane between the origin and the bulk of the data. OCSVM makes theoretical sense when the observations in the lifted space, $\mathcal{F}$, close to the origin are abnormal (and those far away from the origin are normal). Instead of computing transformations into the higher dimensional space explicitly, kernels can be used, since the optimization problem depends only on $k(\boldsymbol{x}, \boldsymbol{y}) := \phi(\boldsymbol{x})^\intercal \phi(\boldsymbol{y})$, a measure of the similarity between transformed samples. Because this function of sample similarities often comes with computation savings compared to explicitly transforming the observations with ϕ, the utilization of the kernel function is called the *kernel-trick*. Possible choices of k are the linear kernel,

$$k(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{x}^\intercal \boldsymbol{y} + c$$

the Gaussian kernel,

$$k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\gamma \|\boldsymbol{x} - \boldsymbol{y}\|^2\right)$$

or sigmoid kernel,

$$k(\boldsymbol{x}, \boldsymbol{y}) = \tanh(\alpha \boldsymbol{x}^\intercal \boldsymbol{y} + c)$$

among others. Following the optimization of the following quadratic program:

$$\min_{\boldsymbol{\alpha}} \frac{1}{2} \sum_{ij} \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}, j)$$
$$s.t.\ 0 \leq \alpha_i \leq \frac{1}{\nu n}$$
$$\sum_i \alpha_i = 1$$

there ends up being a decision boundary at $\sum_i \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) - \rho = 0$, and the outlier score is

$$score_{OCSVM}(\boldsymbol{x}) = -\left(\sum_i \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) - \rho \right)$$

A global hyperparameter, ν, represents the minimum fraction of support vectors (the samples with $\alpha_i > 0$), and maximum fraction of outliers in the training set. Other hyperparameters are specific to each kernel type.

Isolation Forest (IForest) aims to build models of the outliers explicitly [26]. Isolation Forest is an ensemble method - the scores of multiple models are averaged to get the score of a record. Each model is an iTree - a proper binary tree (one where every node has either 0 or 2 children), and each node has an attribute q, along with a test value p. Starting with the training dataset $\boldsymbol{X}$ at the root node, the selection of the q'th dimension is made at random, and p is selected randomly in between the maximum and minimum of the q'th feature in $\boldsymbol{X}$. With part of X now in the left daughter node (those records with $\boldsymbol{x}_q < p$), and the remainder in the right node, and the iTree grows until every child leaf contains a single record per node. When multiple trees are constructed this way, anomalous observations tend to be those that traverse a shorter distance before being isolated. Meanwhile, normal observations tend to be deeper in the tree, since more divisions are needed to separate it from its neighbours.

1. Given $\boldsymbol{X}$, which may be a random subset of the full training data, construct m iTrees

2. For any record, calculate its score on each tree, $d(\boldsymbol{x})$: the depth of traversal before reaching a leaf node

3. Use the negative average depth as the outlier score.

$$score_{iForest} = -\frac{1}{m}\sum_{i}^{m} d(\boldsymbol{x})$$

The usage of these outlier detection algorithms for detecting the onset of machine failure was previously evaluated by Riazi *et al.* [35]. Moreover, a comparative study of the algorithms outlined above was completed, with deep learning outlier detection methods (a subclass of reconstruction-based OD), using different feature construction methods (no transform, PCA, T* [12], and tsfresh [10]), and found no statistically-significant difference in performance between the methods [36]. However, the experiments were set up in a way that does not utilize the temporal aspect of machine condition monitoring data, which we will do.

2.2.2 Outlier Detection for Temporal Data

Gupta et al. [17] surveyed the literature for the methods and applications of outlier detection in time series data. Compared to multivariate techniques, temporal data has significantly more structure that can usually be exploited to increase the performance on a task - whether it be detecting anomalies, or making future predictions. Some types of temporal data are, in order of complexity, are given below. Note that these are not necessarily mutually exclusive.

1. Univariate time series - data collected from a single sensor
2. Multivariate time series - data collected from multiple sensors
3. Data streams - online data (where data is constantly being recorded), and models and algorithms can be constantly self-updating
4. Spatially distributed data - data with an element of spatial correlation and continuity
5. Temporal networks - graph data, usually for data about communications and connectivity

Multivariate outlier detection techniques can be applied to multivariate time series data, by using the sensor value(s) arriving at each time step as a single observation. This is equivalent to partitioning the time series at every time step. When the process supplying the data is periodic, the time series can instead be partitioned at the end of each period. This would significantly increase the number of observations for outlier detection, especially if the period contains many time steps. Doing either of these processes discards some of the advantage that could be gained by knowing the data is temporally correlated.

What can be gained by using time-series specific methodologies? Chandola et al. [8] divides outliers into three classes:

1. Type I - An individual outlying instance w.r.t. the entire data set
2. Type II - An individual outlying instance that is normal w.r.t. the entire data set, but is outlying in its context
3. Type III - A subset of instances that are outlying w.r.t. the entire data set

CBM is targeted towards a mixture of Type II and Type III outliers. Firstly, Type II outliers require either temporal or spatial variables to provide context. By using time series outlier detection techniques, these outliers can be better isolated. Using multivariate methods, they may be completely overlooked. Then, there are Type III outliers, such as the previously discussed *innovative outliers*. These are dependent on the idea of a larger temporal scale, where each individual element may not be outlying, but the group of them are shifting away from the distribution of normal data. Previously, we emphasized the importance of machine failure caused by accumulated wear as the core problem of CBM, which fits firmly within the definition of a Type III outlier.

Starting in the 1970s, the most prominent statistical time-series anaysis and forecasting techniques were based upon Auto-Regressive Integrated Moving-Average models (ARIMA) [46]. These models started off as univariate time-series analysis techniques, but have been adapted to the multivariate case. Relevant to this study is the idea of a *(weakly) stationary time-series*, defined

as a time series $\{\boldsymbol{x}_t\}$ where a constant length sub-time-series $x_t, x_{t+1}, ..., x_{t+k}$ has the same distribution independent of t.

However, around the 1980s, with the introduction of the Kalman filter and linear state-space models, state-space analysis became more prominent [46]. ARIMA has no concept of state; state being the underlying nature of the world (or system) generating the data. State is a latent variable, and the observations are generated as a linear function of state. Linear state-space models also add on to ARIMA modelling with the addition of regressor (a.k.a. input) variables, and native support for the multiple-output (a.k.a. multivariate) case. The discrete linear state-space model (with no input variables) is:

$$\begin{aligned}\boldsymbol{s}(t) =& \boldsymbol{P}\boldsymbol{s}(t-1) + \boldsymbol{e}(t)\\ \boldsymbol{x}(t) =& \boldsymbol{H}\boldsymbol{s}(t) + \boldsymbol{\epsilon}(t)\end{aligned}$$

Here we can see the unobserved state vector, $\boldsymbol{s}$, and a state-to-state linear transition function $\boldsymbol{P}$. The observation vector $\boldsymbol{x}(t)$ is a linear function of state, and both state transitions and observed variables have error terms $\boldsymbol{e}$ and $\boldsymbol{\epsilon}$ respectively. ARIMA and state-space models can be interchanged [46], but the mapping from one form to the other is not necessarily one-to-one. [1]

This state-space model of the world highlights the Markov property: that the current state fully defines the distribution of the next state. A system that satisfies this Markov property is memory-less, formally described as

$$\mathbb{P}(S_t \mid S_{t-1}) = \mathbb{P}(S_t \mid S_{t-1}, S_{t-2}, S_{t-3}, ..)$$

Moreover, this allows as to speak even more generally about non-linear state-space models, namely, Hidden Markov Models (HMM):

$$\begin{aligned}\boldsymbol{s}(t) =& \boldsymbol{f}(\boldsymbol{s}(t-1)) + \boldsymbol{e}(t)\\ \boldsymbol{x}(t) =& \boldsymbol{g}(\boldsymbol{s}(t)) + \boldsymbol{\epsilon}(t)\end{aligned}$$

where $\boldsymbol{f}$ and $\boldsymbol{g}$ can be either linear or non-linear functions.

2.2.3 Markov Chain for Outlier Detection

A Markov Chain is like a HMM, except that observations are a *known* function of state. Therefore using Markov Chains for outlier detection relies on the as-

sumption that the states are fully observable using the sensor values (observations). Let there be n multivariate time series ($\boldsymbol{X}_i \in \mathbb{R}^{T \times k}$, $i \in \{0, 1, ..., n-1\}$), with constant period T and with k sensors.

1. Sort sensor values into contiguous bins, where the bins are of equal width and cover the entire range of the sensor/feature.

2. For the observations at each time step, map the observation $\boldsymbol{x} \in \mathbb{R}^k$ back into the state $s_t \in \mathcal{S}$. The discrete state space $\mathcal{S}$ is the Cartesian product of the k binnings[4]. *E.g.* if there are 4 bins per sensor, and 3 sensors, $\mid \mathcal{S} \mid = 4^3$

3. Construct the starting probability vector $\boldsymbol{u}$

$$u_i = \text{number of times a time series starts with } S_i$$

4. Construct the transition probabilities $\boldsymbol{P}$, empirically:

$$P_{ij} = \frac{\text{number of times where } S_j \text{ follows } S_i}{\text{number of times where } S_i \text{ occurs}}$$

5. Given any observation $X \in \mathbb{T}^{T \times k}$, find its state space representation (in $\mathcal{S}^T$), and use the negative log likelihood as its outlier score

$$score_{MC} = -log(u(s_0)) - \sum_{t=0}^{T-1} log(P(s_t, s_{t+1})))$$

2.3 Reinforcement Learning

Reinforcement Learning (RL) [42] is the problem of learning about, navigating, and controlling an environment, often in an online and incremental manner. Central to the RL problem is the Markov Decision Process (MDP). Furthermore, the RL methods we introduce in this section are adapted for the development of GVFOD. There are two intertwined goals of RL: prediction, and control. In this thesis, we focus only on RL for prediction, in which case the MDP effectively degenerates into a Markov Reward Process (MRP).

[4]We make the assumption that this state space $\mathcal{S}$ is the true state space behind the Markov Chain.

2.3.1 Markov Decision/Reward Processes

The Markov Decision Process is a model of an agent's interactions with the world. The MDP is defined with a state space $\mathcal{S}$, an action space $\mathcal{A}$, the transition measure $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}^+$, and the reward measure $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathbb{R} \mapsto \mathbb{R}^+$.

Based on the current state, $S_t \in \mathcal{S}$, the agent will take an action sampled from its policy, $A_t \sim \pi_t = \mathbb{P}(A_t \mid S_t)$. Afterwards, the environment will inform the agent of its reward R_{t+1} and next state S_{t+1}. Thus the chain of experience is

$$S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, ...$$

The behaviour of the agent can be stochastic. Likewise, the dynamics of the stochastic environment determine the next state and reward based upon the current state and action: [5]

$$S_{t+1} \sim \mathcal{P}(S_t, A_t, S_{t+1}) = \mathbb{P}(S_t \mid S_{t-1}, A_{t-1})$$
$$R_{t+1} \sim \mathcal{R}(S_t, A_t, R_{t+1}) = \mathbb{P}(R_t \mid S_{t-1}, A_{t-1})$$

This demonstrates the Markov property - where the transitions from the current state are fully determined and independent of prior states.

The difference between prediction and control in RL is whether the agent modifies its behaviour π as experience is gathered. In control, the policy is modified over time in order to maximize future rewards. In the prediction task, estimates of the *value* of a state are made given a policy π.

The *return* is defined as a exponential weighted sum of future rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...$$

The discount factor, γ, lies in the range $[0, 1)$, in order to ensure the return remains finite. When $\gamma = 0$, the value is *myopic*, and the return is equal to the next reward. This return is stochastic: it depends on the stochastic starting state, S_t. Even controlling for the stating state, there's still randomness in the

[5] For simplicity, we introduce these with conditionally independent S_{t+1} and R_{t+1}

future rollouts of experience. Thus, the expected return as a function of state is desired, named the *state-value function*, is desired:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

The state-value function depends on the random distribution of the chain of experience. In the case that the policy remains constant, we can remove the *decision* aspect of an MDP, leaving a Markov Reward Process (MRP). We can disregard the idea of actions, and marginalize it into the dynamics of the environment.

At this point[6], the only difference between the MRP and a Hidden Markov Model is the observability of state. Further discussion on observability is left to the section on function approximation.

2.3.2 Temporal Difference Learning

Temporal difference (TD) learning aims to generate estimates of the state-value function. One of the driving principles behind TD learning is that it is online and incremental, meaning that learning is completed after each piece of data arrives, and discarded after. This ensures that the algorithm scales well - and learning and acting do not slow down as additional experience accrues. This limitation can be a technical challenge. An algorithm that can learn value functions by referring to a stored database of MRP experience cannot necessarily learn online, at least not in a computationally efficient manner.

TD learning makes updates to its value function based on the TD error (δ, TDE), and an estimate of the value function ($\hat{v}$)

$$TDE = \delta = R_{t+1} + \gamma \hat{v}(S_{t+1}) - \hat{v}(S_t)$$

The first part, $R_{t+1} + \gamma \hat{v}(S_{t+1})$, is the estimate of the value function using the latest experience R_{t+1}, S_{t+1}, while the second term is the previous value estimate. The difference in estimated value with the passage of time gives way to the naming *temporal difference*.

[6] If we also assume the conditional independence of S_t and R_t given S_{t-1}

To reduce noise in the estimates, the state value functions are updated in small increments: $\hat{v}(S_t) \leftarrow \hat{v}(S_t) + \alpha\delta$. The value function estimates take a small step to correct the TD error, with a step-size $\alpha \in (0, 1]$.

2.3.3 Function Approximation

The *tabular* setting of TD learning described above updates only states that are visited. However, if the state space is very large, each state will be visited so seldomly that it would be unreasonable to expect learning to occur very quickly. In that case, generalization is required to learn approximate value functions over the entire state space. This is also true for continuous state spaces - where the number of individual states is uncountably infinite. Using *function approximation*, updates to a value function will affect not only the visited state, but also nearby states.

A special case of function approximation is the linear form:

$$v(s) \approx \hat{v}(s) = \boldsymbol{w}^\intercal \boldsymbol{\phi}(s)$$

Here, the feature vector $\boldsymbol{\phi} : \mathcal{S} \mapsto \Phi \subseteq \mathbb{R}^d$ is a mapping from state space to the feature space. This mapping needs to be cleverly constructed in a way that allows for appropriate generalization (learning on one state improves nearby states) and resolution (the differences in value between nearby states can be represented). If the state space is a subset of $\mathbb{R}^{|\mathcal{S}|}$, then common feature mappings are polynomial, Gaussian, and Fourier basis functions.

An effective and computationally efficient feature mapping is tile coding [41]. In this representation, the feature vector is sparse and binary ($\boldsymbol{\phi}(s) \in \{0, 1\}^d$), and the number of non-zero elements is always constant. Tile-coding maps multidimensional continuous spaces from the observation space into a sparse binary feature vector. There is one active feature per tiling - corresponding to the tile that the observation lies in (see Figure 2.2). For a 2-D bounded continuous input space, we can imagine a grid of unit squares (tiles), with the corners aligned to integer values. Additional tilings can be added that are offset from one another and the main tiling. With n tilings, the feature vector is a binary vector with n ones, each indicating the location of the

active tile for each tiling. This method of feature construction allows for both generalization and specificity of the value function estimates. The amount of generalization is defined by the size of the tiles - in Figure 2.2, the value function estimates are updated for all the states within the 4 thick bounded squares. Meanwhile, specificity can still be achieved; the intersection of those 4 boxes is the smallest portion of the observation space that must share the same value estimate.

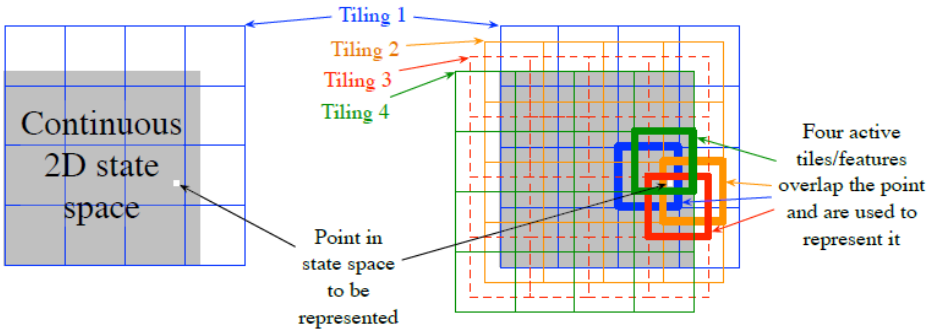


Figure 2.2: A demonstration of Tile Coding. The gray area is the bounded and continuous observation space. A number of offset tilings are , the active tile in each of the tilings composes the feature vector. With 4 tilings, there will always be 4 active features in the binary feature vector. Source: Sutton and Barto [42].

In order to learn, the weights $\boldsymbol{w}$ get modified through stochastic gradient descent. This is known as the TD(0) algorithm with linear function approximation:

$$\begin{aligned}\boldsymbol{w} \leftarrow &\boldsymbol{w} + \alpha \left[R + \gamma \hat{v}(s') - \hat{v}(s)\right] \nabla_w \hat{v}(s) \\ = &\boldsymbol{w} + \alpha \delta \boldsymbol{\phi}(s)\end{aligned}$$

In order to learn a little bit more quickly, updates need not be made only to the most recently visited state and its neighbours. A history of recently visited states can be kept in a memory-efficient fashion, using an *eligibility trace* $\boldsymbol{z} \in \mathbb{R}^d$. With each step, the entire trace decays geometrically with decay factor λ, and then the current representation $\boldsymbol{\phi}(s)$ is added. The full

TD(λ) algorithm, with function approximation, is given as:

$$\boldsymbol{z} \leftarrow \gamma\lambda\boldsymbol{z} + \boldsymbol{\phi}(s)$$
$$\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\delta\boldsymbol{z}$$

Up until now, reinforcement learning has been presented for solving MDPs or MRPs. In an MDP, the true state is visible, and value updates are made to value functions for the state; with function approximation, updates are also made to nearby states. More often than not, the true state of the environment is not observable by the agent, and it receives an approximation of state: the observations $\boldsymbol{x}$. In an industrial machine, these observations come from sensors, corrupted by noise, and almost certainly fails to uniquely represent the entire state of the system. This partial observability is characterized in a POMDP - Partially Observable Markov Decision Process. This is analogous to the relationship between a Markov Chain and a Hidden Markov Model.

In practice, the fact that TD(λ) was developed assuming full observability of state is of little concern. Linear function approximation limits the space of representable state-value functions, analogous to how observations are approximations of state. Therefore, although the theoretical rigour is lost by using observations instead of true state, empirical evaluations can still be done to see if the methods are effective. Another commonly discussed topic is agent-state: where the observations are sufficient for the agent to build a representation of state. Further discussion of observability and state can be found in Section 17.3 of Sutton and Barto [42].

Henceforth we will simply use the observations as an approximation of state: $\boldsymbol{x} \approxeq s$

2.3.4 General Value Functions

A general value function extends the idea of a state-value function beyond estimating discounted sums of rewards. Instead, any signal can be predicted. A signal can be external - as in sensor values, or it can be internal - like model parameters, errors and their statistics etc. A non-reward signal like this is

called a *cumulant*, $C_t \in \mathbb{R}$. The general value function to given as

$$v_C(s) = \mathbb{E}\left[\sum_{k=t}^{\infty} \gamma^{t-k} C_{k+1} \mid S_t = s\right]$$

and this can be estimated using linear function approximation as well:

$$v_C(s) \approx \hat{v}_C(s) = \boldsymbol{w}_C^\intercal \boldsymbol{\phi}(\boldsymbol{x})$$

General value functions serve as predictions. When $\gamma = 0$, it's a myopic prediction of the expected next signal. Chaining myopic GVFs (making predictions of predictions) allows for a prediction some number of time steps into the future [44]. With $\gamma \in (0, 1)$, it is a prediction of the discounted sum of the cumulant, just as in a value function. The discount factor γ can be a function of state as well: $\gamma(S_t)$. One example is a robot exploring a room - where its cumulant signal is 1 if the robot moves in one direction, and -1 if it moves in the opposite direction. By setting γ as 1 normally, and 0 when the agent reaches a wall, the general value function provides an estimate of the distance in that direction to a wall.

GVFs were used in an industrial setting to form predictions of a welding process [16]. GVFs were able to predict the quality of a weld seam using a feature representation combining both tile-coding and a deep-learning autoencoder. Predictions at different time scales were made by modifying γ. A *Horde* [43] of GVFs can create predictions in the world directly, in order to extract as much knowledge as possible. These GVFs can further be used as a basis to build agent-state for the value function. These individual GVF predictions are called *demons* - a name chosen in reference to the Pandemonium architecture. These predictions were scaled up in Modayil et al. [29], and demonstrated the ability of TD learning to learn 6000 GVFs using a reasonable amount of computational hardware. This usage of GVFs imitates the biological phenomena of *nexting* - the "propensity of people and many other animals to continually predict what will happen next." [29]

These predictions effectively form a model of the world - any divergence from this model would be surprising, and perhaps an indicator of something

gone wrong. This was introduced as "unexpected demon error", or UDE, in White et al. [49]. Günther et al. [15] demonstrated that UDE can effectively detect unexpected disturbances to a machine, and keep records of which perceptions were affected by a surprising event. GVFOD will adapt these techniques to the semi-supervised learning problem of machine failure detection - using GVFs as a predictive model of the world, and an adaptation of UDE to create the outlier score.

2.4 Testbench Construction

To generate a machine failure dataset, a robot arm was commissioned at the University of Alberta, Department of Mechanical Engineering. The details of the robot arm's hardware, software, and dynamics are described here. This section is a summary of the content in [36] and [25].

2.4.1 Hardware

The robotic arm apparatus (Figure 2.3) consists of two major components - the arm and the driver. The arm itself is supported on one end by a bearing, fixed in position, but permitted to rotate about the vertical axis. The other end of the arm is supported by an aluminum wheel and bearing, which rolls on a flat steel plate. A holder for removal weights is located on the arm; for all experiments, a stack of plates weighing 5.5kg was added to this arm.

The driver is a belt and pulley system. There are three pulleys, and three belt-segments between them (Figure 2.4). The three pulleys are named as the *motor pulley*, the *idler pulley*, and the *arm pulley*. The arm pulley is attached to the arm. The idler pulley position is movable, by adjusting the 4 tensioner screws that keep it in position. The motor pulley is attached to a shaft, driven by a motor. The shaft has a torque sensor embedded within. The belt looping around these three pulleys is contiguous with the exception of a cut made in Belt Section 3 - between the motor and arm pulley - where a tension meter / strain gauge was installed. The belt and pulley profiles are toothed - no appreciable skipping was noted through the entire data collection

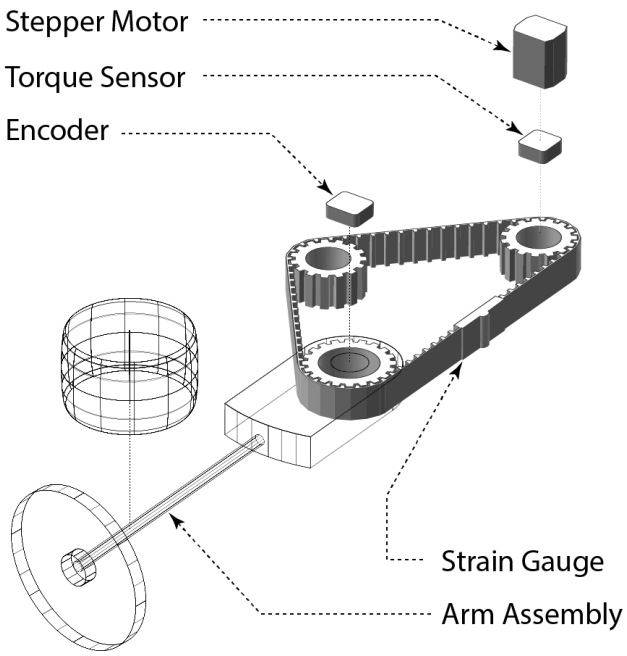


Figure 2.3: Robot arm hardware.

process. The nominal belt tension is $160N$. The entire driver is rigid and firmly mounted to a 3 tonne seismic platform to reduce vibrations.

2.4.2 Software

In operation, the arm regularly moves between two [nominal] angular positions: 19° and 126°. Three sensors record data: an encoder on the arm pulley, recording the arm position; a torque sensor on the motor output shaft; and a tension sensor on the belt between the motor and arm pulleys. The sampling rate of all sensors is 200Hz. The velocity profile of the arm is approximately trapezoidal. With a period of $10s$, the arm takes $5s$ to move from 19° to 126° and pauses, and takes the remainder of the $5s$ to move and pause.

2.4.3 Empirical Fault Data

The amount of data collected for normal and faulty operation is given in Table 2.1

The overwhelming majority of data was collected under normal operating conditions. In order to generate faulty data, the machine was modified in a way that simulates operation in a near-failure condition. This included modifying

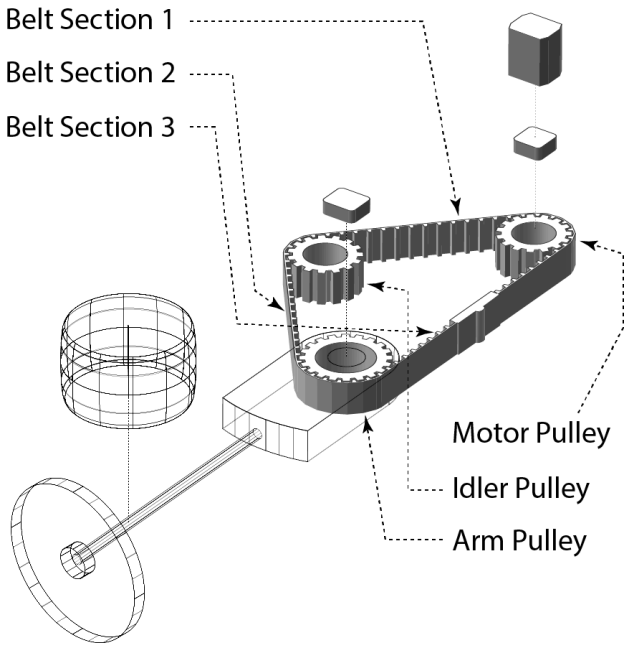


Figure 2.4: Robot arm component names.

the belt tension with the tensioner on the idler pulley; with addition of sand onto the steel plate upon which the arm rolls; and with heating the system with a floodlight.

Table 2.1: Robot arm fault dataset.

Failure Type	Samples	Nominal Belt Tension (N)	Description
Normal (non-faulty)	7193	160	Normal operation.
Tension - Loose L2	180	120	Varying belt tension levels.
Tension - Loose L1	182	140	
Tension - Tight	194	180	
Sandy	183	160	Sand on track rolling surface
High Temperature	210	180	Robot arm heated with two incandescent floodlights. Surface temperature ∼40 °C

2.4.4 Dynamical System Modelling

Lipsett et al. [25] derived a dynamical model of the robot arm. A diagram of this is given in Figure 2.5. The model models the dynamics of the three pulleys

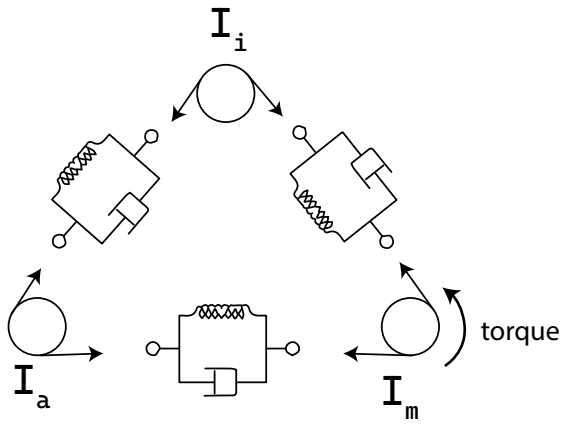


Figure 2.5: The dynamical system model of the robot arm given in Lipsett *et al*. [25].

as rigid bodies that are free to rotate. Each pulley has two parameters: radius and moment of inertia. The belts are modeled as Kelvin-Voigt viscoelastic elements - with a single spring and dashpot in parallel. Although the actual apparatus consists of one contiguous belt, it is modelled as three separate belt sections. Each belt section has a different length l_i, and their dampening factors d_i and stiffnesses k_i are a function of this length. These three belt sections share a damping factor C_L and stiffness EA[7]. The belt specific spring constant and dampening constants are given by:

$$k_i = \frac{EA}{l_i} = \frac{\text{Young's Modulus} \times \text{Cross Sectional Area}}{\text{Length}}$$
$$d_i = \frac{C_L}{l_i} = \frac{\text{Damping Factor}}{\text{Length}}$$

In the modelling and equation, we use the following conventions:

- The positive sense is CCW
- The subscripts m, i, a denote motor, idler, and arm respectively
- The belt sections are:

    1. between motor and idler

[7] EA is the product of Young's modulus E and the cross-sectional area A; these were not measured independent, but the product of them was empirically known

2. between idler and arm

3. between arm and motor

At all times, the state is fully defined by the angular position (θ) and angular velocity ($\dot{\theta}$) of each of the three pulleys.

The tension in each belt is a deterministic function of the belt properties and the pulley position and speed. As an example, the tension in Belt Section 1 is given as:

$$F_{t1} = max\left(0, F_{t,base} + k_1(\theta_i r_i - \theta_m r_m) + d_1(\dot{\theta}_i r_i - \dot{\theta}_m r_m)\right)$$

where T_{base} is the static equilibrium tension in the belt. Furthermore, the differential equations describing the dynamics of the pulleys are:

$$\begin{aligned}
\ddot{\theta}_m I_m &= r_m(F_{t1} - F_{t3}) + \tau_m \\
\ddot{\theta}_i I_i &= r_i(F_{t2} - F_{t1}) \\
\ddot{\theta}_a I_a &= r_a(F_{t3} - F_{t2})
\end{aligned}$$

where τ_m is the torque applied to the motor pulley by the motor shaft.

These equations, and knowledge of all the system properties (such as belt lengths, pulley radii, belt properties, etc.), constitute the dynamical model of machine behaviour in Lipsett *et al.* [25]. The *input* of this dynamical system is the motor torque, τ_m, and the *output* is the position and velocities of each pulley; i.e., the state vector. For the system to be solved, the initial state also needs to be defined. If all these components are available, any generic differential equation solver can find the machine state as a function of time and motor torque input.

Chapter 3

General Value Function Outlier Detection - GVFOD

In this chapter, we introduce the GVFOD algorithm. Furthermore, we evaluate its performance compared to multivariate techniques on the robot arm fault dataset.

3.1 GVFOD - General Value Function Outlier Detection

The motivation behind GVFOD is to learn general value functions as a predictive model of the machine's normal behaviour, and detect outliers using these models.

The training of GVFOD takes place in two stages. First, the algorithm makes a single forward pass through the normal training data to learn a group of GVFs. Afterwards, the models are saved, learning is stopped, and a second pass through the training data is made to learn the distribution of errors. These errors are calculated using the fixed model. Future data can be evaluated based on its deviation from the learned models. In order to determine if the machine has entered a faulty operating condition, the prediction errors from the testing data is compared to distribution of errors learned in the training phase. In doing so, we have transformed the online nature of GVFs and TD learning into an offline method, but retain the incremental-learning aspect.

To construct the group of GVFs (*Horde* of *Demons*), the cumulants need

to be chosen. Choosing which predictions each GVF needs to make can be difficult; for machine fault detection, we suggest that there be one GVF per sensor, with the sensor value as the cumulant, and one shared discount factor (γ) for all of them. Since the GVF estimates the discounted sum of each cumulant into the future, the discount rate sets a pseudo-horizon for the sum of ($\frac{1}{1-\gamma}$).

General value functions are functions of state. In practice, the true state of a machine is never completely observed. The only visible variables are the observations - which themselves are some unknown function of state, measurement noise, system inputs etc. Using these observations $\boldsymbol{x}$ as the *agent-state*, features can be constructed, through some mapping from observation space to feature space. These feature functions can be as simple as using the observations themselves (using an identity map). When this is insufficient, the true GVFs will not be well represented as a linear cannot of these features, and a more complex feature representation may be needed. Previous study of temporal difference learning has found that tile-coding [42] is effective. Open-source implementations of this include *tiles3* [41] and *tile_coder* [14].[1]

$$v_{C_i}(s) = \mathbb{E}\left[\sum_{k=t}^{\infty} \gamma^{t-k} C_{k+1} \mid S_t = s\right] \approx \boldsymbol{w}^\intercal \boldsymbol{\phi}(\boldsymbol{x}) = \hat{v}_{C_i}(\boldsymbol{x}(s))$$

With a list of cumulants C_i, a discount rate γ, and a method to transform observations into features, the value function is defined and can be estimated. The step-size α and trace-decay parameter λ are hyperparameters that need to be set prior to learning. Using linear function approximation, the $TD(\lambda)$ algorithm can estimate the GVFs. After one pass through the training data, the GVF parameters $\boldsymbol{w}_{C_i}$ are saved, and constitute the model of machine behaviour. In a second pass, the Unexpected Demon Error (UDE) [49] is calculated using this fixed model and the temporal difference errors δ_i .

$$UDE(t) = surprise(t) = \left| \frac{\frac{1}{\beta}\sum_{i=t-\beta}^{t} \delta_i}{s_{\delta,0:t}} \right|$$

[1] *tiles3* uses hashing - often resulting in slower runtimes, but less memory usage. *tile_coder* does not utilize hashing, which increases the memory requirements, but is vectorized to take advantage of modern processor architectures

The standard deviation of δ_i is calculated using only historical TD errors: $s_{\delta,0:t} = \sqrt{var\{\delta_0, ..., \delta_t\}}$; in an online learning setting, UDE could be calculated online as well. β is a hyperparameter - corresponding to the *window* over which TD errors are averaged.

In cases where an outlier score is desired for a multivariate time series $\boldsymbol{X} \in \mathbb{R}^{T \times d}$, there will be T UDEs for each of the d cumulants. The average UDE can be used as the outlier score.

$$score_{GVFOD}(\boldsymbol{X}) = \frac{1}{T \times k} \sum_{t=1}^{T} \sum_{i=1}^{k} UDE_i(t)$$

GVFOD is not a multivariate outlier detection algorithm for the following reasons:

1. GVFOD requires the input data to be time series. For a multivariate OD algorithm, shuffling the features for all samples (*e.g.* shuffling the order of observations in a single period) would have no effect. Doing the same shuffling for GVFOD would likely reduce its performance.

2. Likewise, the training samples need to remain ordered, whereas a multivariate OD algorithm can learn from shuffled training data. The training of GVFOD by $TD(\lambda)$ with a constant step-size is intentional. Samples later on in the training set have a larger effect on the GVF parameters than earlier samples, and the learned from earlier samples can be overwritten.

3. Testing data cannot be shuffled, since the outlier scores are not independent. This is caused by UDE being a function of not just the current TD error, but also of previous errors. In the experimental section of this chapter, GVFOD's testing set always starts with normal data, and is followed by fault data. This non-standard arrangement of test data is valid, since a real machine would be expected to be working normally before it encounters a fault.

The full algorithm is given in Algorithm 1.

Algorithm 1 GVFOD (for a single cumulant)

Input:

Training samples $\mathcal{D}^{train} = \{\boldsymbol{X}_i, \boldsymbol{c}_i\}_{i=1}^{n}$,
$\boldsymbol{X}_i \in \mathbb{R}^{T \times d}$, $\boldsymbol{c}_i \in \mathbb{R}^T$
Function approximator $\phi : \mathbb{R}^d \mapsto \mathbb{R}^k$
Contamination ratio $\alpha \in (0, 0.5)$

Input:

Testing samples $\mathcal{D}^{test} = \{\boldsymbol{X}_j, \boldsymbol{c}_j\}_{j=1}^{m}$

Output:

Outlier scores $\boldsymbol{y}^{test} \in [0, \infty)^m$
Outlier threshold $y^* \in [0, \infty)^m$

Initialize:
$\boldsymbol{w} \in 0^k$, $\boldsymbol{z} \in 0^k$
$\alpha > 0$, $\lambda \in [0, 1]$, $\gamma \in [0, 1)$, $\beta \in \mathbb{N}$
for $\boldsymbol{X}_i, \boldsymbol{c}_i \in \mathcal{D}^{train}$ **do**
  for $t = 0$ to $T - 1$ **do**
    $\delta \leftarrow C_{t+1} + \gamma \boldsymbol{w}^\intercal \phi(\boldsymbol{x}_{t+1}) - \boldsymbol{w}^\intercal \phi(\boldsymbol{x}_t)$
    $\boldsymbol{z} \leftarrow \gamma\lambda \boldsymbol{z} + \phi(\boldsymbol{x}_t)$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\delta \boldsymbol{z}$
  end for
end for
$\boldsymbol{w}_C = \boldsymbol{w}$
for $\boldsymbol{X}_i, \boldsymbol{c}_i \in \mathcal{D}^{train}$ **do**
  for $t = 0$ to $T - 1$ **do**
    $\delta_t \leftarrow C_{t+1} + \gamma \boldsymbol{w}_C^\intercal \phi(\boldsymbol{x}_{t+1}) - \boldsymbol{w}_C^\intercal \phi(\boldsymbol{x}_t)$
    $UDE_t \leftarrow surprise(\boldsymbol{\delta}_{0:t}, \beta)$
  end for
  $y_i \leftarrow \frac{1}{T} \sum_{t=0}^{T-1} UDE_t$
end for
$y^* = quantile(OS, (1 - \alpha))$
for $\boldsymbol{X}_j, \boldsymbol{c}_j \in \mathcal{D}^{test}$ **do**
  for $t = 0$ to $T - 1$ **do**
    $\boldsymbol{\delta}_t \leftarrow C_{t+1} + \gamma \boldsymbol{w}_C^\intercal \phi(\boldsymbol{x}_{t+1}) - \boldsymbol{w}_C^\intercal \phi(\boldsymbol{x}_t)$
    $UDE_t \leftarrow surprise(\boldsymbol{\delta}_{0:t}, \beta)$
  end for
  $y_j^{test} \leftarrow \frac{1}{T} \sum_{t=0}^{T-1} UDE_t$
end for
return $\boldsymbol{y}^{test}, y^*$

3.2 Experiment

In this application, each sample of robot arm data is a flattened multivariate time series. Each sample has 6000 features. This corresponds to the 3 time series (position, torque, tension), each with a length of $T = 2000$. A large challenge with outlier detection, and machine learning in general, is the curse of dimensionality. In order to reduce the dimensionality, principal components analysis (PCA) can be used to produce a lower dimensional representation of the data. PCA is optimal in the sense that it retains the maximum possible variance in the data using a limited number of variates. PCA is sensitive to feature scaling; if PCA is used, each of the 6000 features is scaled to zero mean and unit variance beforehand.

In this experiment, the number of principal components used is 20, which retains 96.2% of the variance in the normal data (Figure 3.1). Whether or not PCA is used before passing data to the outlier detection algorithm is a hyperparameter that will be chosen for each algorithm.

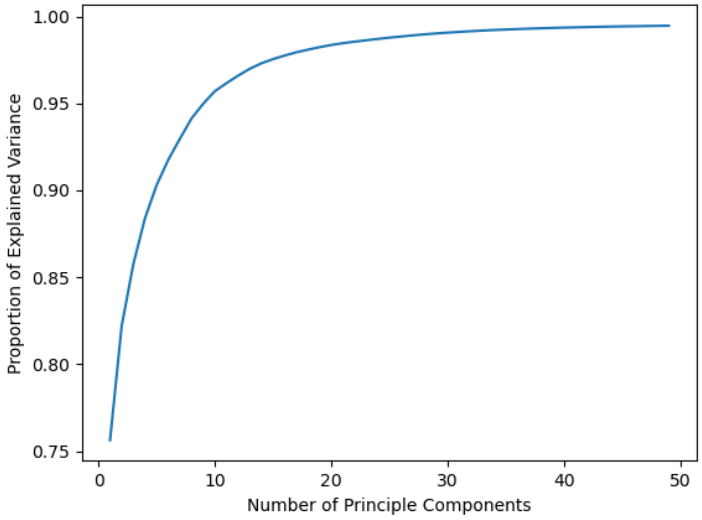


Figure 3.1: The proportion of explained variance is shown as a function of the number of principal components retained. Features were scaled to unit variance and zero mean prior to application of PCA. Standard multivariate analysis techniques require that the number of retained components exceeds the *elbow* on the graph. We determine visually that this elbow occurs around $n = 12$. This graph is truncated at $n = 50$, but the plot reaches 1 at $n = 6000$.

After the preprocessing step, each algorithm is evaluated by training on normal data, and then evaluating on a test dataset of normal and abnormal data. The following metrics were used to evaluate performance. We maintain the convention that positive observations are faults, and negative observations are normal records.[2]

$$precision = \frac{tp}{tp + fp}$$
$$recall = \frac{tp}{tp + fn}$$
$$F1 = \frac{2}{recall^{-1} + precision^{-1}}$$

For every algorithm, hyperparameters need to be set before the training phase. The only shared parameter between all outlier algorithms is the contamination ratio. The contamination ratio determines the cutoff between normality and abnormality - it is a quantile of the outlier scores on the training data. In all experiments, the contamination ratio was set to 5%. Thus, any test record with an outlier score greater than 95% of the training scores will be classified as an outlier.

For every other hyperparameter, Hyperopt [3] was used to find the optimal value. *Optimal* is used very loosely here - there is no guarantee that the selected parameters are the best; it is limited by the search algorithm, the computational resource availability, and the number of evaluations completed.

Optimization was done using only *hyperparameter tuning* data from the first 12.8 hours of normal data and the first half of each fault dataset. This segmentation was done in order to keep a set of fresh data for evaluating the algorithms afterwards (see Figure 3.2). The last 12.8 hours of normal data, and last half of each fault dataset, is termed the *model evaluation* set or *validation* set

The optimal parameters found for each algorithm is given in Table 3.1. The optimization algorithm used was Tree of Parzen Estimators [2]. The sequential model-based optimization is done as follows:

[2] tp is the number of true positives, fp are the false positives, tn are the true negatives, and fn are the false negatives.

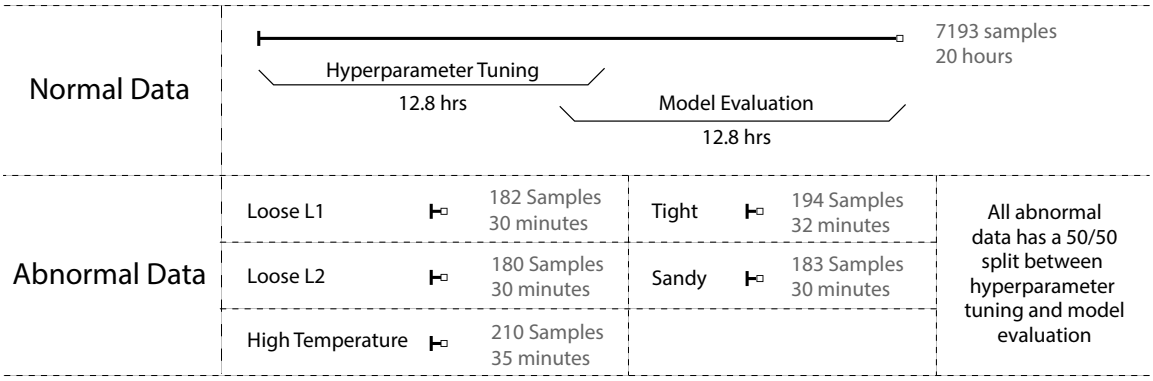


Figure 3.2: A visualization of the utilization of the normal and fault data. The first 12.8 hours of normal data are used for hyperparameter tuning, and the last 12.8 hours of normal data are used in algorithm evaluation. To achieve the desired sample sizes there is some overlap. Fault data is split 50/50 with no overlap.

1. A candidate set of hyperparameters is chosen by Hyperopt.
2. Repeat the following 10 times, using the splitting scheme shown in Figure 3.3.
   (a) Get the training set of 1000 contiguous normal records. Train the OD algorithm with the selected hyperparameters.
   (b) Use the subsequent 1000 contiguous normal records, as well as the abnormal data, as the testing set. Get outlier scores, evaluate against the outlier threshold, and get F1-score.
3. Report the average F1-score of these 10 runs back to Hyperopt.
4. Repeat the above 400 times. Return the best hyperparameters.

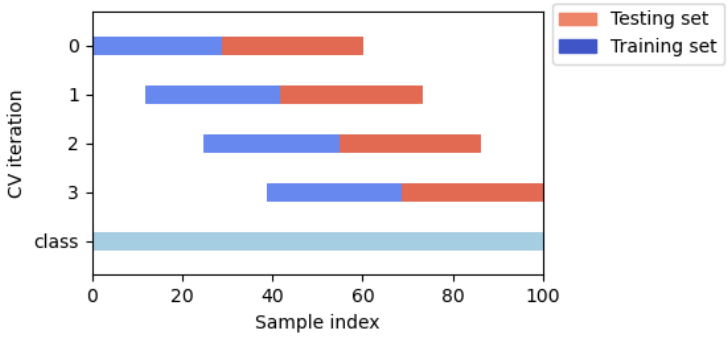


Figure 3.3: The method by which normal data is split for hyperparameter optimization. In practice, 10 splits are made, with 1000 periods in training, and 1000 periods in testing. Abnormal data is not split - since it is only ever used in testing.

Table 3.1: Algorithms and parameters used.

Algorithm	**Class**	**PCA**	**Hyperparameter**	**Default Value**	**Optimal Value (on this dataset)**
IForest	Domain-based	False	n_estimators	100	14
			max_features	1.0	0.88
			bootstrap	False	True
OCSVM	Domain-based	True	kernel	rbf	sigmoid
			nu	0.5	0.996
			gamma	1/n_features	4.280e-05
			coef0	N/A	0.103
LOF	Distance-based	True	n_neighbors	20	500
			metric	Euclidean	Chebyshev
ABOD	Distance-based	True	n_neighbors	5	92
kNN	Distance-based	True	n_neighbors	5	500
			metric	Euclidean	Chebyshev
HBOS	Probabilistic	False	n_bins	10	10
			alpha	0.1	0.827
			tol	0.5	0.754
MCD	Probabilistic	True	support_fraction	0.5	0.714
PCA	Reconstruction	False	n_components	All	3
			weighted	True	False
			whiten	False	True
Markov Chain	Temporal	False	divisions	N/A	8
GVFOD	Temporal	False	divs_per_dim	N/A	[7, 7, 2]
			numtilings		2
			discount_rate		0.96
			learn_rate		0.104
			lambda		0.209
			beta		888

3.3 Results and Discussion

Algorithm evaluations were done with a single outlier class at a time. In general, *Loose L1* data exhibited the worst performance, and are reported here. The 95% confidence interval on the mean is displayed in every plot. Performances on other outlier classes are provided in Appendix A.

The metrics used to evaluate algorithm performance are precision, recall, and F1-score, all of which have been previously defined. These metrics have good intuitive meaning when applied to machine fault detection. An algorithm with low recall will fail to identify machine faults when the behaviour is truly faulty. An algorithm with low precision will wrongly classify normal operation data as faulty. Ideally both would be high - and F1-score is a harmonic mean of the two. Additionally, all these metrics are sensitive to the balance of normal vs abnormal classes. To ensure that different experiments can be appropriately compared, the number of samples in the normal and the *Loose L1* abnormal class are kept constant in all test sets.

The metrics are calculated as a function of the training data used, since it's worth knowing the effect that training data quantity has on algorithm performance. It is obviously beneficial if an algorithm can learn with a smaller amount of data, since there would be a shorter lead time between machine startup and the availability of the fault detection algorithm. Moreover, if machine behaviour is non-stationary in its normal operation, it would be interesting if additional training causes deteriorated performance. To evaluate this, the splitting scheme for each run is shown in Figure 3.4. Splitting is only done for normal data - since abnormal data only goes into the testing set. In order to have an estimate of error, there are offsets of 10 minutes for each experiment, and a total of 20 experiments. *I.e.* the last experiment starts at minute 200. In each experiment, there are a further 20 runs, each taking a different amount of training data. The runs range from between 50 samples (8m20s) and 2000 samples (5h33m) of training data; the subsequent 600 samples (1h40m) of normal data, and *Loose L1* outlier data is used for testing. Note that these experiments are not independent, and that there is less inde-

pendence between runs at large training sizes: the offsets between runs stays constant, but the amount of overlapping data grows.

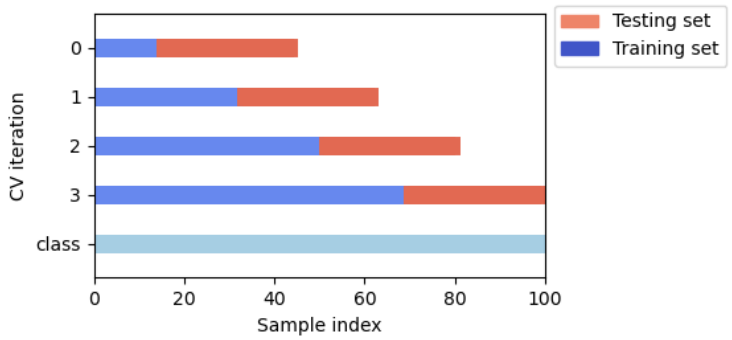


Figure 3.4: Normal data splitting method. In this study, 20 runs are made, with anywhere 50 to 2000 periods in training, and 600 periods in testing. Abnormal data is not split - since it is only ever used in testing.

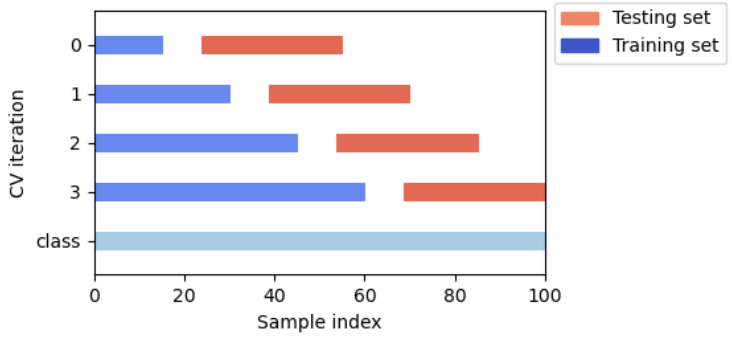


Figure 3.5: Normal data splitting method. A delay is added between training and testing sets.

Figure 3.6 shows the performance of different algorithms on the *hyperparameter tuning* data. Since we optimized the hyperparameters on this data, this plot represents the best-case performance. However, even though these results cannot be reported as the algorithm's testing performance, it's still worth investigating whether there is a difference in performance between training and testing sets. In general, it can be seen that most algorithms have very good recall - indicating that most algorithms are capable of identifying faulty behaviour. The algorithm performance is better discriminated by their precision. In general, GVFOD outperforms all the other algorithms, with no significant

change in performance from training sizes between 600 and 2000. Below a training size of 600, the precision of most algorithms is poor. Curiously, the performance of all other algorithms seems to drop off as training data increases. More importantly, we need to evaluate these algorithms on validation data - data which was [mostly] unused for hyperparameter tuning.

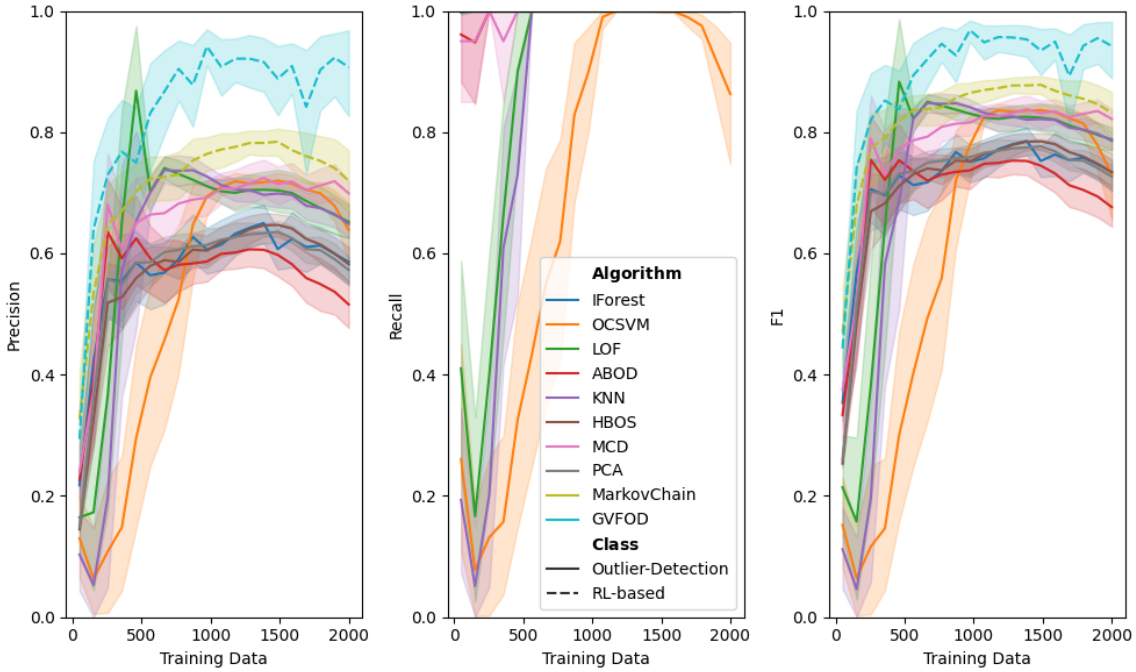


Figure 3.6: Algorithm performance on "hyperparameter tuning" normal data and Loose L1 fault data. Optimal parameters used.

Figure 3.7 shows the performance using the *validation* data. Overall, we see a significant decrease in model performance compared to Figure 3.6. When encountering reduced model performance on validation data, a likely cause that needs to be ruled out is overfitting - where the hyperparameters have been selected to fit the noise of the *hyperparameter-tuning* data. Another plausible explanation is that in the *validation* data the underlying mechanisms generating machine data are shifting significantly, and the normal test data is being sampled from a process which has changed since learning on the training data. Of note is also the drop in performance for most multivariate OD algorithms and Markov Chain as training data is increased from 500 periods to 1500 periods. In general, more data leads to better performance, but that

is clearly not the case here. The exception is GVFOD, which seems to scale linearly with more data, and hits 100% precision and recall at a training size of 2000. These promising results come with two caveats: there is more overlap in the training sets as training data size increases, leading to less independence between runs. Additionally, GVFOD tends to have greater variance than most other methods. This can be a large problem in practice, since this can manifest as inconsistent algorithm performance.

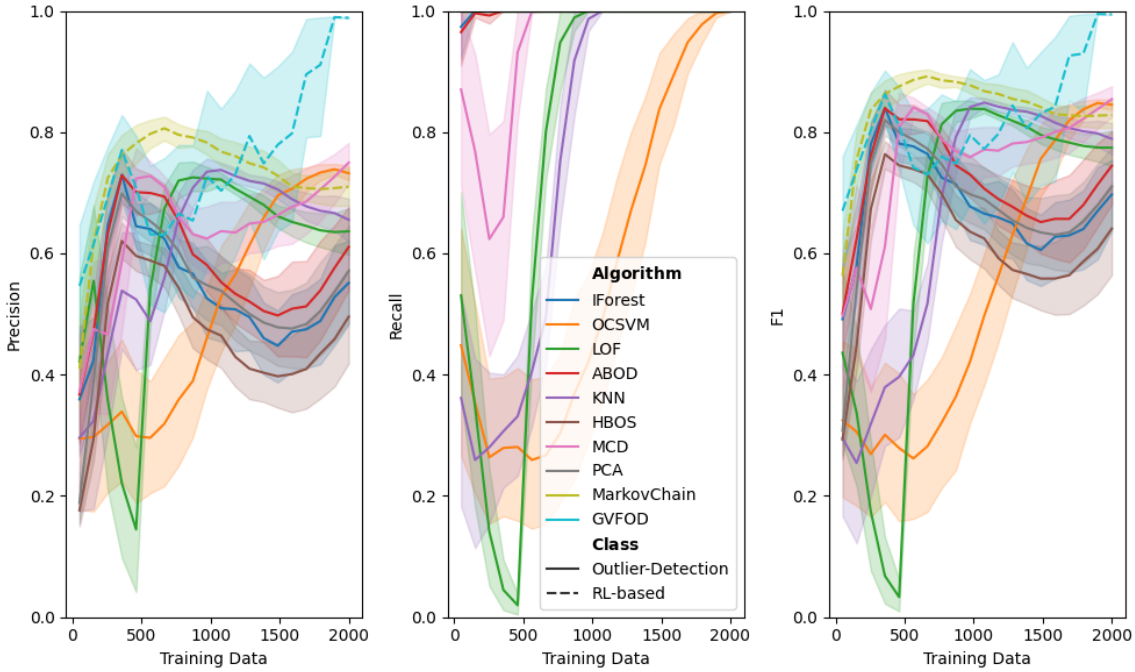


Figure 3.7: Algorithm performance on "model evaluation" normal data and Loose L1 fault data. Optimal parameters used.

Furthermore, Figure 3.8 shows experiments that are done with a 720 period (2 hour) delay between training and testing, using the validation data. In a machine fault detection setting, this would be similar to the operational case where the model was trained using older data, and has not been updated with the most recently available *normal* data. The splitting scheme which incorporates a delay can be seen in Figure 3.5. The difference between performance with and without delay is likely due to non-stationary behaviour in the robot arm data. If the normal data arising from the robot arm was identically distributed over time, there would be no such reduction in performance

when a delay is added between testing and training. Hence, we conclude that there exists significant non-stationary behaviour in the robot arm data; furthermore, it seems that GVFOD is suitable for learning normal behaviour in non-stationary environments, shown by its consistent performance.

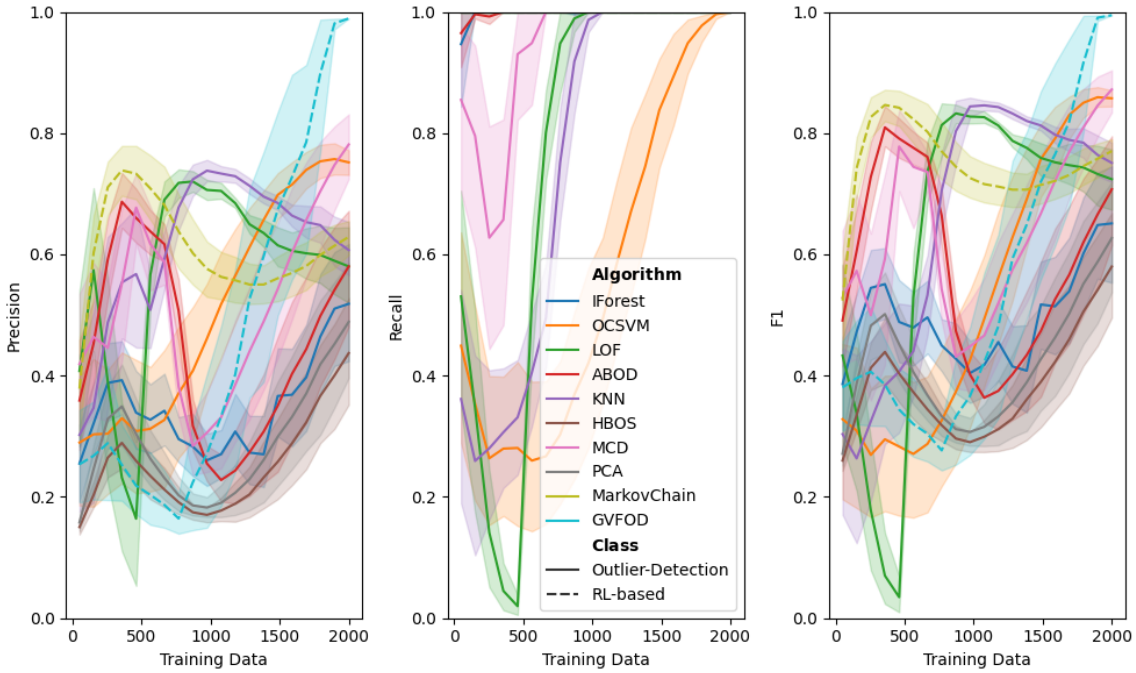


Figure 3.8: Algorithm performance on "model evaluation" normal data and Loose L1 fault data. Optimal parameters used. A two hour delay occurred between training and testing.

It would be remiss not to mention Markov Chain performance in Figures 3.7 and 3.8: similar to GVFOD, Markov Chain is a state-space model of time series outlier detection. It may not have the same asymptotic performance that GVFOD has exhibited, but it consistently performs about as well as other multivariate techniques. Moreover, it has significantly less variance than GVFOD - a highly desirable property in practice.

In a machine fault detection application, fault data is often not available. Imagine, after installing a brand new machine, that it has to be modified to simulate failure. This would seem like a ridiculous action, and it would probably reduce the machine's reliability in the future. As such, we evaluated the performance of these algorithms using the default values provided

in PyOD (Table 3.1). For GVFOD, no default parameters are available, so parameters were chosen based upon general knowledge of the machine and of reinforcement-learning.

1. *divs_per_dim*: the number of tiles (or bins) to have for each dimension (sensor). The chosen value is $[10, 10, 10]$, which corresponds to the number of tiles for *position*, *torque*, and *tension* respectively. This determines the amount of generalization: learning on one state can at most affect value function estimates of states $\frac{1}{10} = 10\%$ of the sensor range in either direction.

2. *numtilings*: the number of tilings to use. It also determines the number of ones in the feature vector. The chosen value is 10. Combined with the previous choice of *divs_per_dim*, there is $\frac{1}{10\times10} = 1\%$ resolution in each dimension.

3. *discount_rate*: The chosen value is 0.9. This yields a pseudo-horizon of the GVF predictions of $\frac{1}{1-0.9} = 10$ time-steps; and the discounted sum (and the value function) will be on a scale roughly 10x the scale of the cumulant.

4. *learn_rate*: the amount of correction with each visit to a state. The chosen value is 0.01. This value needs to be in the range $(0, 1]$. Note that when using the TD-learning rule, $\alpha = \frac{learn\_rate}{numtilings}$, to ensure there is no overshoot in the updates. If there is no noise, and the true value function is fully representable with the feature mapping, the ideal value is always 1. The inverse of *learn_rate* is similar to the time-constant - the "elapsed time required for the [error] to decay to zero if the system had continued to decay at the initial rate" [45]. With a *learn_rate* of 1%, the error would decay to zero in 100 visits to that state if the initial rate of learning was kept.

5. *lambda*: the trace-decay parameter. The chosen value was 0.1. This value is typically fairly arbitrary - larger values pass more information

backwards in time to update previously visited states, but have higher variance.

6. *beta*: the "window width" for UDE. The chosen value was 250. This should be set to a value representative of how many time steps it would take to recognize an outlier. Since the period of the robot arm is 2000 steps, a choice of 250 seemed reasonable.

These choices of hyperparameters were made without any feedback from the algorithm performance, in order to ensure there is no unintentional tuning of results, which was done previously using parameter optimization. We wish to conduct this experiment in a way to best mimic a real deployment of a fault detection algorithm, where no fault data is available for tuning the hyperparameters, and therefore there would be no objective function to optimize over the hyperparameters.

Markov Chain's hyperparameter, *divisions*, was kept the same as the optimal value. The meaning behind that parameter is the same as GVFOD's *divs_per_dim*, except that this implementation of Markov Chain requires that all sensors must be divided into the same number of tiles.

The counterparts to the previous three experiments, using these default hyperparameters, are shown in Figures 3.9, 3.10, and 3.11. Again we see better performance on the *hyperparameter tuning* data than on the *model evaluation* data. Since parameter search was not done, this is evidence against overfitting in the first set of experiments, although it does not rule it out completely. However, the performance drop could be much better explained by changes in the machine - indicating there is significant non-stationarity in normal machine behaviour. Interestingly, the expert-selected GVFOD parameters are better than what Hyperopt could find in 400 trials. Recall that the SMBO procedure searched for the best average F1-score at a training size of 1000 samples; at this training size, the optimal parameters acheived a mean F1-score of only 96.8% compared to the expert-selected parameter F1-score of 98.8%. Most other algorithms behaved as expected, where SMBO finds better hyperparameters than the default values. Using parameter optimization, LOF, kNN, OCSVM,

and MCD exceed an F1-score of 80% somewhere along the curve; without optimization, only MCD exceeds 80%.

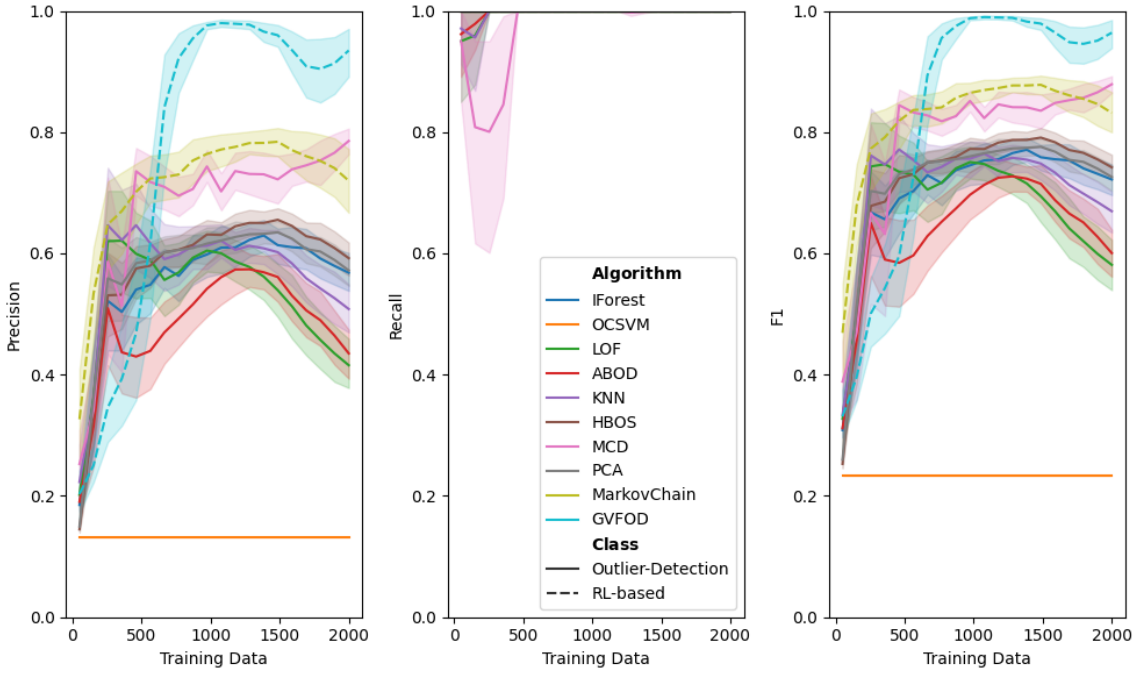


Figure 3.9: Algorithm performance on "hyperparameter tuning" normal data and Loose L1 fault data. Default parameters used.

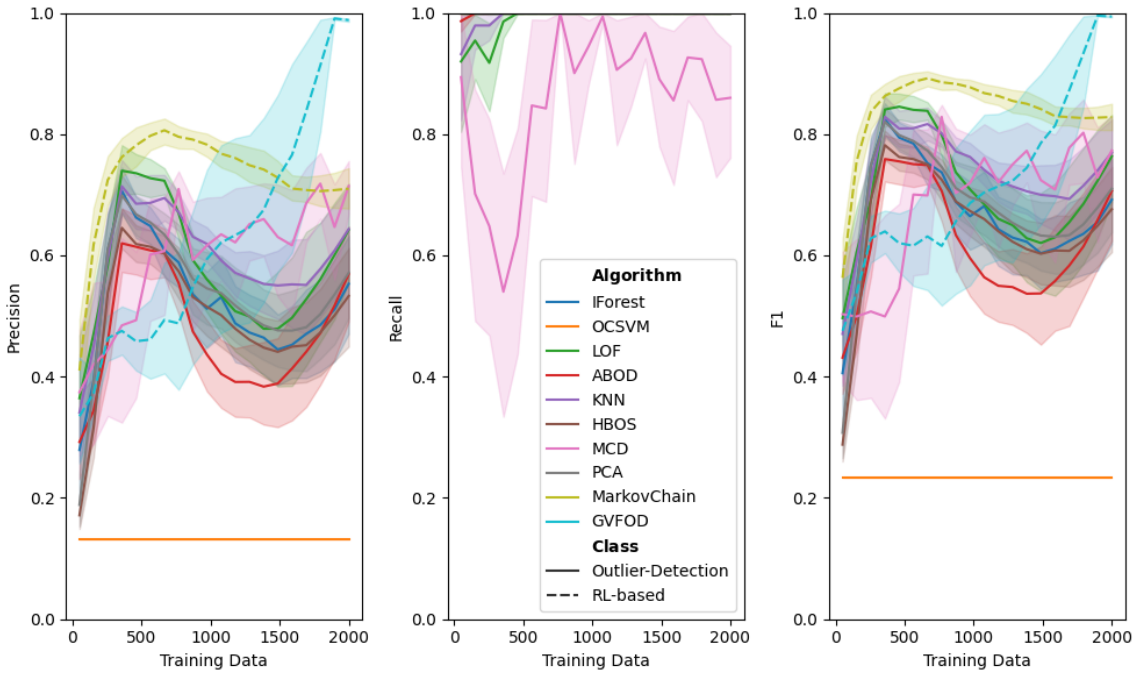


Figure 3.10: Algorithm performance on "model evaluation" normal data and Loose L1 fault data. Default parameters used.

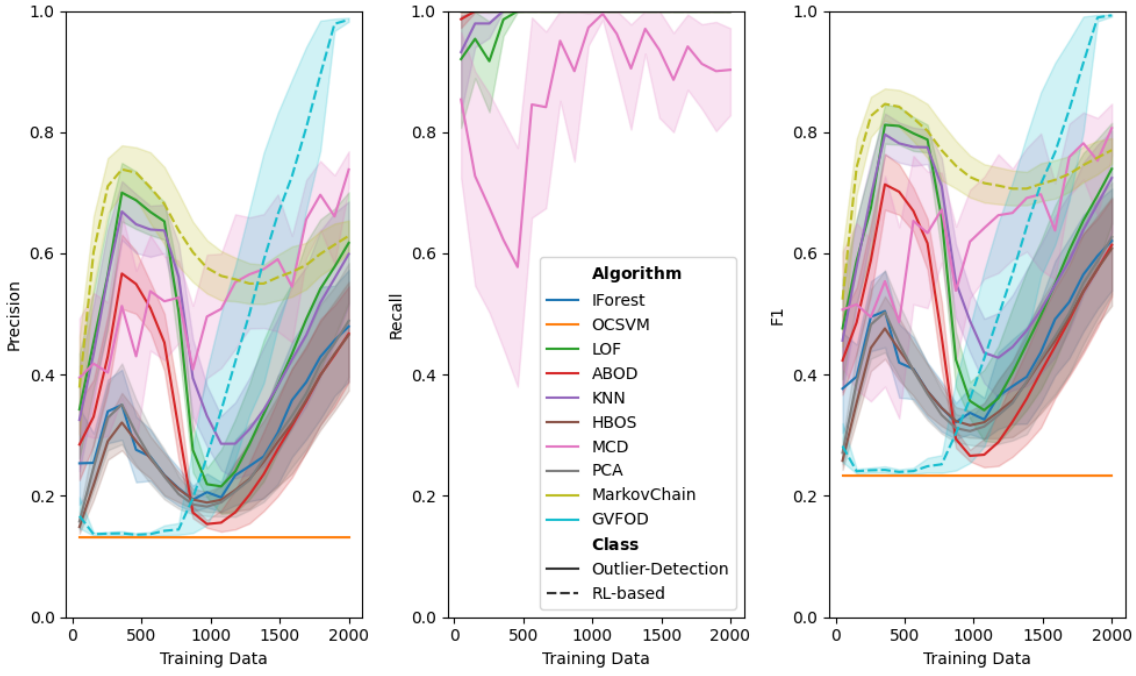


Figure 3.11: Algorithm performance on "model evaluation" normal data and Loose L1 fault data. Optimal parameters used. A two hour delay occurred between training and testing.

From these experiments, we have identified low precision as the biggest limitation of machine learning algorithms for fault detection in the robot-arm dataset. High recall in a machine fault detection setting is important - it indicates that a machine can reliably identify faulty behaviour when it occurs. However, with the number of false positives reported, any of the multivariate outlier detection algorithms would quickly overwhelm an operator with machine-fault warnings, rendering the fault detection system near-useless.

Kay *et al.* [22] discusses the importance of tying computing science measures of algorithm performance (precision, recall, accuracy, etc.) to the acceptability of the algorithm in a *user-facing* application. In this case, a single period of machine operation takes $10s$. If an algorithm achieves a 99% false positive rate, there will still be 86 false alarms per day! In an industrial setting, this would have to be scaled up for each machine in the fleet. It's not difficult to imagine that there would be very little user trust in the CBM system.

In practice, we recommend using averaging techniques or other low-pass filtering techniques to remove the noise from the outlier scores. With the requirement that CBM targets only accumulated-wear failures, it is a valid assumption that only when the *average* rate of fault alarms rises, does there exist a fault in the system behaviour.

3.4 Conclusion

In this chapter, we introduced General Value Function Outlier Detection (GVFOD), a new algorithm for detecting outliers in machine operation data. GVFOD uses general value functions as a predictive model of machine behaviour, and Unexpected Demon Error (UDE) as a measure of *surprise*. GVFOD was compared to Markov Chain and a selection of multivariate outlier detection algorithms for finding faulty behaviour in the robot-arm dataset. Using the Tree of Parzen Estimators [2] algorithm for sequential model-based optimization, we searched for the best parameters for every algorithm using both normal and faulty data. GVFOD performed at least as well as other algorithms in the majority of situations. With the availability of more training data, GVFOD

seemed to outperform all other algorithms. Furthermore, GVFOD parameters were selected using expert knowledge, without the use of fault data; it was found that the intuitive nature of GVFOD parameters for time series data led to good performance relative to other algorithms. Some drawbacks of GVFOD include its reduced performance at small training sizes compared to multivariate outlier detection methods, and its high variance overall. Lastly, we discovered significant non-stationarity in the latter part of the robot arm dataset, within the *normal* operating condition. Since this non-stationarity is uncontrolled, the next chapter will build a simulator for further evaluation of GVFOD in synthetic non-stationary environments. Overall, these contributions to the field of CBM highlight the importance of exploiting the temporal dimension of data for machine fault detection, and how developments in online-and-incremental RL methods can achieve equal-or-better performance in offline semi-supervised learning problems.

Chapter 4

Fault Detection for Simulated Non-Stationary Machine Behaviour

In this chapter, we introduce an updated simulator of the robot arm, and use it to generate data for outlier detection. In particular, we are able to simulate gradually changing machine conditions in a controlled manner. We then compare LOF, GVFOD, and UDE for the purpose of fault detection in CBM.

4.1 Robot Arm System Identification

First, we need to evaluate the performance of the simulator in Lipsett *et al.* [25]. We pick a small two-period (20s) slice of operation early on in the normal dataset, starting at period 500 (1h23m since machine start). The initial conditions are calculated under the assumption that the robot arm starts off stationary, and with no net force on any pulley. Using linearly interpolated torque data as the input, we solve the equations given in Subsection 2.5 using a numerical differential equation solver. The solver used is LSODA [40], implemented in the SciPy package.

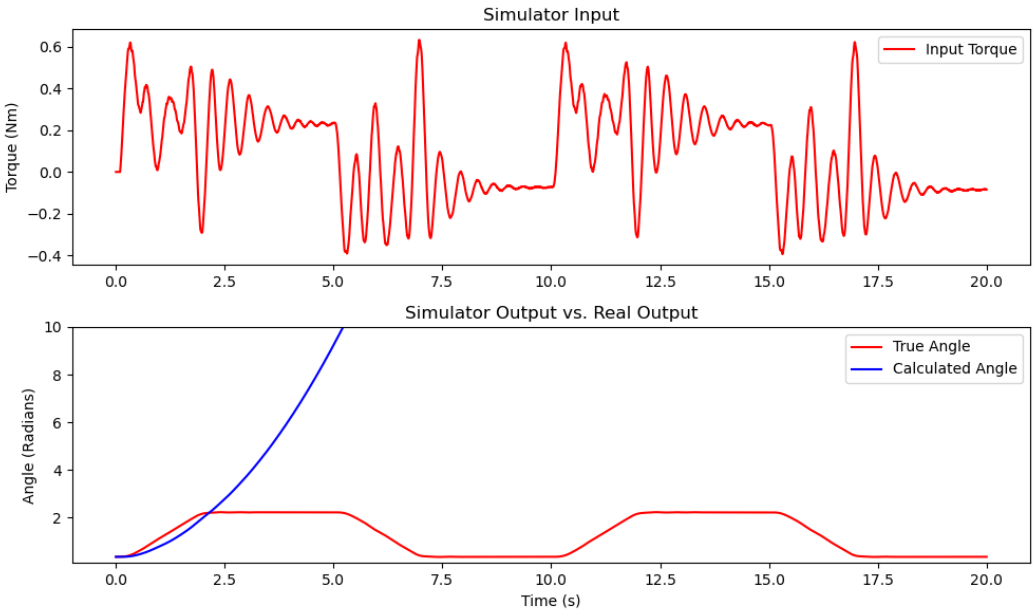


Figure 4.1: Performance of default parameters on robot-arm data.

Figure 4.1 shows the simulated performance of the robot arm using the default parameters. The calculated angle diverges from the true angle almost immediately. The dynamical model of the simulated robot arm is therefore not representative of the real arm. These default parameters were empirically measured and derived [25], and are tabulated in Table 4.1. Two major considerations need to be made:

1. The only source of energy dissipation in the default model is through belt damping parameters. Since the arm angle increases in a superlinear fashion, there likely needs to be more sources of energy dissipation through friction.

2. The torque applied at 5s and the torque applied at 10s are unequal and non-zero. However, at those times, the arm is nearly stationary. Therefore, there needs to be an equal and opposite force elsewhere in the robot arm system. A likely source is that the surface upon which the arm rolls is not perfectly level, and the arm is applying a torque to the arm pulley as it rests on the slanted surface.

To address these issues, additional forces are added to the system. Friction

forces need to be added to each of the three pulleys. Adding friction can be difficult - the Coulomb model of friction is a non-differentiable function of the tangential force, the normal force, and the pulley speed. Instead, we implement a linear model of friction, similar to a viscous model, and attach an additional soft-sign term. An example of this frictional force for the motor pulley is given as:

$$\tau_{f,m} = -N_m(f_{1m}\,\mathrm{sgn}(\dot{\theta}_m) + f_{2m}\dot{\theta}_m)$$

where N_m is the normal force applied to the pulley by the two belt sections. f_{1m} is the frictional term similar to the Coulomb model of dynamic friction, except it is implemented with a soft-sign (sigmoidal) function[1] for differentiability, which is necessary for the initial value problem solver used. The second term, f_{2m} corresponds to the viscous model of friction - where the frictional force is proportional to the speed of the pulley. A similar model of friction is used on the other pulleys.

In order to account for the force of gravity and the slope in the rolling surface, an additional force is added to the arm pulley:

$$\tau_g = slope_1 \times \sin(\theta_a) + slope_2 \times \cos(\theta_1)$$

Altogether, the new equations defining the motion of the robot arm are:

$$\begin{aligned}
\ddot{\theta}_m I_m &= r_m(F_{t1} - F_{t3}) + \tau_m(t) + \tau_{f,m} \\
\ddot{\theta}_i I_i &= r_i(F_{t2} - F_{t1}) + \tau_{f,i} \\
\ddot{\theta}_a I_a &= r_a(F_{t3} - F_{t2}) + \tau_g + \tau_{f,a}
\end{aligned}$$

To learn all the parameter values, the same Sequential Model-Based Optimization (SMBO) scheme is used: Tree of Parzen Estimators [2], which is implemented in the Hyperopt package [3]. The optimal values are shown in Table 4.1. Although this package was developed for optimizing hyperparameters in computer vision architectures, it proves to be useful here. Certain parameters (C_L, EA) have a large error between empirical and optimized values, due to insensitivity of the simulator's arm position to the parameter value.

[1] $\mathrm{sgn}(x) = \tanh(20x)$

The objective function was the mean-squared-error (MSE) between the calculated and measured arm angle, over the same $20s$ of data in Figure 4.1. All 20 parameters were allowed to be modified; however, certain parameters that have a lower probability of measurement error (such as belt lengths, pulley diameters) were restricted to a neighborhood of their empirical values. The size of the neighborhoods was chosen as $\pm 5\%$ for belt lengths and $\pm 25\%$ for pulley diameters. 20,000 runs were completed to find these parameters, taking roughly two days on 16 Broadwell Xeon cores.

Table 4.1: Optimal and empirical parameters for robot arm simulator.

Parameter	Description	Optimal Value (Hyperopt)	Laboratory Measured Value
C_L	Belt damping factor	$5.506Ns$	$0.183Ns$
EA	Belt stiffness constant	$41079N$	$7058N$
l_1	Length of belt section 1	$0.1263m$	$0.127m$
l_2	Length of belt section 2	$0.1529m$	$0.152m$
l_3	Length of belt section 3	$0.1787m$	$0.178m$
I_m	Mass moment of inertia (motor pulley)	1.51×10^{-4} $kg \cdot m^2$	unknown
I_i	Mass moment of inertia (idler pulley)	3.99×10^{-7} $kg \cdot m^2$	2.445×10^{-6} $kg \cdot m^2$
I_a	Mass moment of inertia (arm pulley and assembly)	0.300 $kg \cdot m^2$	0.547 $kg \cdot m^2$
r_m	Radius of motor pulley	$0.0115m$	$0.0122m$
r_i	Radius of idler pulley	$0.0103m$	$0.0122m$
r_a	Radius of arm pulley	$0.0209m$	$0.0189m$
T_{base}	Base tension in belt	$155N$	$155N$
f_{1m}	Friction parameter 1 (motor pulley)	$7.3 \times 10^{-7}m$	unknown
f_{2m}	Friction parameter 2 (motor pulley)	$5.2 \times 10^{-8}s$	unknown
f_{1i}	...	$1.3 \times 10^{-5}m$	unknown
f_{2i}	...	$6.2 \times 10^{-6}s$	unknown
f_{1a}	...	$3.4 \times 10^{-4}m$	unknown
f_{2a}	...	$5.7 \times 10^{-4}s$	unknown
$slope_1$	Slope correction factor 1	$-0.341Nm$	unknown
$slope_2$	Slope correction factor 2	$0.310Nm$	unknown

Figure 4.2 shows the performance using the optimal parameters. The first 20 seconds is the training data - the arm position's MSE was optimized over

this range. The training loss over these 20 seconds was 1.39. In the following 20 seconds, the testing loss was 2.12. The testing loss in the last 20 seconds was 2.27. Initial value problems can be sensitive to the initial conditions. If this is the case here, it may be evidenced by errors that propagate and grow over time. In Figure 4.2, we can see that the MSE grows over the 40 seconds following the training data.

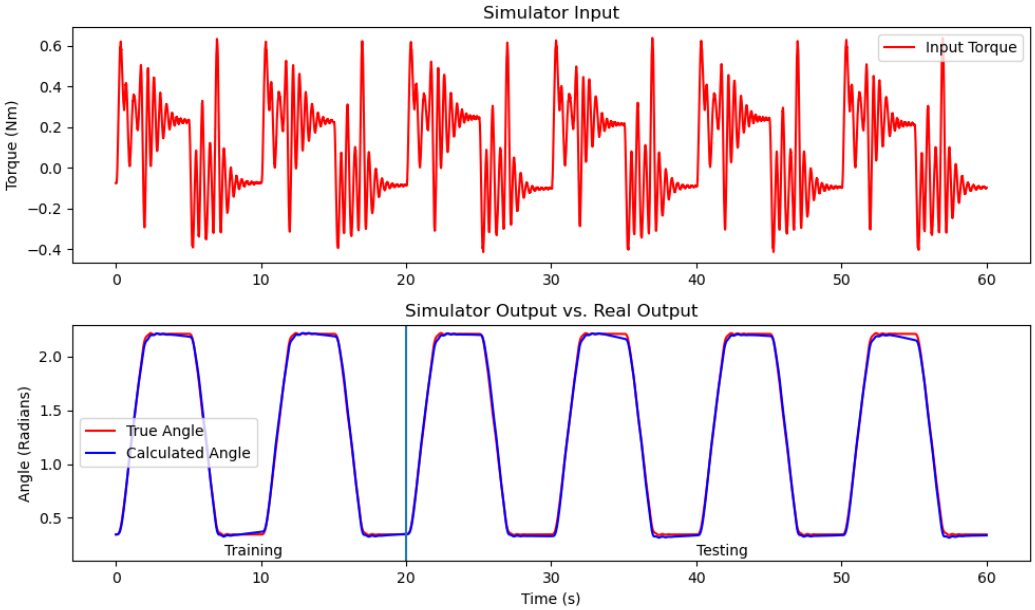


Figure 4.2: Performance of default parameters on robot-arm data.

Furthermore, when the initial motor position was increased by 1% (from $0.533rad$ to $0.539rad$), the loss in the training data increased to 2.17 from 2.12, and the following 20 seconds of data saw the MSE increase from 2.12 to 2.23. We can therefore assume that as time increases, the errors will continue to propagate, and the arm's position will diverge from the measured values. Therefore, we cannot use empirical torque data to generate robot arm data over a significantly longer period of time, since the range of motion will drift over time.

4.1.1 Control

To complete the simulator, and release the dependence on empirical torque measurements, a PID controller was implemented. The process variable to be

Table 4.2: PID control parameters.

Control Parameters					
K_p **Gain**	25 Nm/rad	T_{int} **Integration Time**	2.0 s	T_{dt} **Derivative Time**	0.01 s

controlled is the arm angle, and the manipulated variable is the motor torque.

Ideally, the controller of the experimental robot arm would be ported into our simulation. However, that code is unavailable, and instead we define our own setpoint and PID parameters. We do not seek to match experimental and computed torque profiles, since the purpose of this simulation is to generate normal and abnormal data for the robot arm. What matters is a logical effect of the system dynamics parameters on system response, and not whether the simulated system dynamics match the true dynamics exactly.

A basic way of smoothly moving an object from one point to another is by using a trapezoidal velocity profile (Figure 4.3). The object accelerates up to its maximum speed, maintains this speed, and decelerates such that it reaches 0 velocity at its desired location. By integrating this velocity profile, we have the setpoint for robot arm position. This setpoint has an 'S' shape. The IVP solver used is requires the differentiability of the arm angle setpoint: although the function is defined piecewise, and the trapezoidal velocity profile ensures the setpoint is differentiable.

The PID controller is given as a function of the arm position error, $e(t) = \theta_a^{SP}(t) - \theta_a(t)$, and with motor torque $\tau_m(t)$ as the output.

$$\tau_m(t) = K_p \left(e(t) + \frac{1}{T_{int}} \int_0^t e(t')dt' + T_{dt} \frac{\mathrm{d}e(t)}{\mathrm{d}t} \right)$$

The parameters chosen for this equation are given in Table 4.2. These parameters were selected by hand, to visually minimize the error in the robot arm position. Note that $\frac{d}{dt}e(t) = \frac{d}{dt}\theta_a^{SP}(t) - \dot{\theta}_a(t)$, which can be calculated using the existing state variables. The missing variable, $E \doteq \int_0^t e(t')dt'$ is added to the state vector, using the first order ODE $\dot{E}(t) = e(t)$, with an initial value of zero.

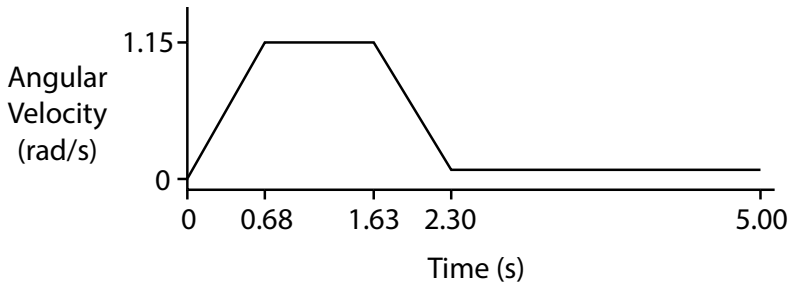


Figure 4.3: Trapezoidal velocity profile of the robot arm simulator target (in CCW direction).

Table 4.3: State vector and initial values of dynamical system.

Parameter	Description	Initial Value
θ_m	Motor pulley angle (rad)	0.533
$\dot{\theta}_m$	Motor pulley velocity (rad/s)	0
θ_i	Idler pulley angle (rad)	0.533
$\dot{\theta}_i$	Idler pulley velocity (rad/s)	0
θ_a	Arm pulley angle (rad)	0.344
$\dot{\theta}_a$	Arm pulley velocity (rad/s)	0
$E = \int_0^t \theta_a^{SP} - \theta_a dt'$	Cumulative arm pulley angle error $(rad \cdot s)$	0

4.2 Experiment

Figure 4.4 demonstrates the ability of this controlled simulator to generate data for the robot arm in perpetuity, at the same sampling rate and with the same sensors/observations as the empirical machine. Because there is feedback control on the robot arm position, the arm position will always tend towards its setpoint. The torque values are given by the PID controller, and the tension is given by the equations in Section 2.4.4. Note that there is a difference between empirical and simulated torque values and tension values. By visual inspection, the periodicity and range of the simulated data seem reasonably similar to empirical values, so we can accept this as a good simulator to base further analysis. We will verify this by modifying parameters and seeing if they have explainable effects on the simulated data. Recall that the goal is not to emulate the behaviour of the robot arm perfectly, but to have a reasonable

model that can generate data with explainable parameters.

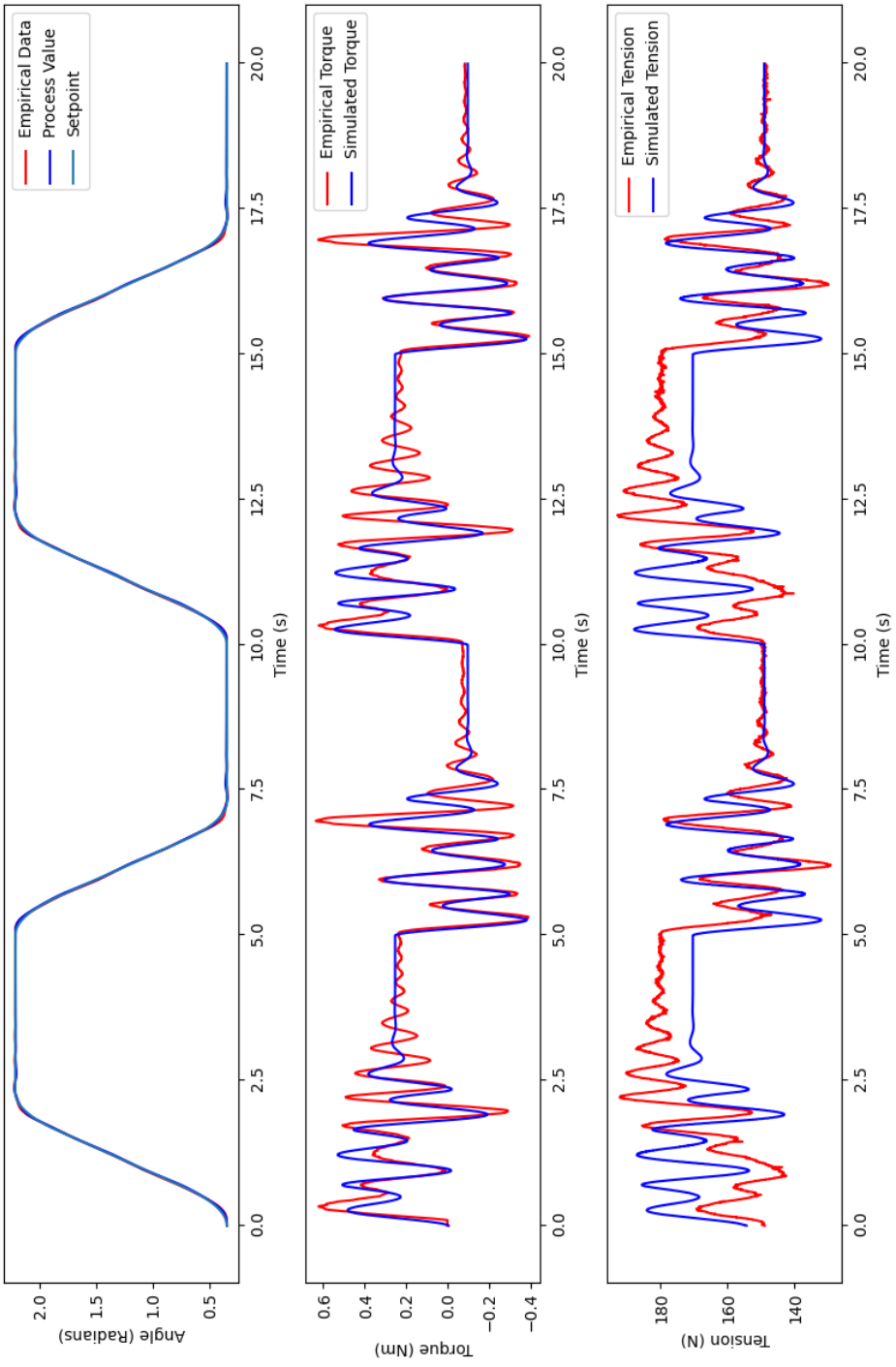


Figure 4.4: Simulated controlled robot arm data. The torque and tension are calculated as functions of state.

The parameters of the model can now be modified to see what changes then induce in the data. This will serve as a sanity check that the simulator

behaves in ways we expect. In Figure 4.5, the static belt tension is modified about its nominal and optimal value of 155N. It can be seen that the arm position remains nearly unchanged (due to the PID control), which will also be seen in the later experiments. Here we also see that the torque magnitude increases when the arm is in motion and when it is being accelerated, but has a smaller magnitude when the arm is being decelerated - this can be explained by increased friction from a tighter belt. The dynamic tension measurements are as expected, the majority of the change being due to the offset in static tension.

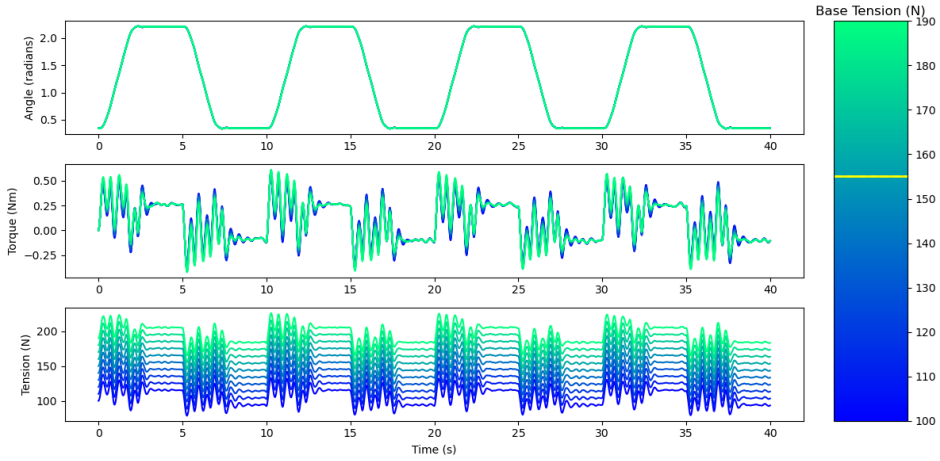


Figure 4.5: Changing the base belt tension ($F_{t,base}$) in the PID controlled robot arm simulator.

In Figure 4.6, the rolling surface of the arm has been tilted in one direction. We would expect that the slope-induced torque in the arm pulley would change as a function of the arm position. We see the effect of this in the motor torque measurements and in the tension measurements - where there is a significantly larger effect when $t = 2.5s, 12.5s, ...$ as compared to when $t = 7.5s, 17.5s, ...$

Lastly, in Figure 4.7, the viscous friction in the arm pulley is being modified. Because viscous friction is proportional to arm velocity, we would expect the biggest changes to arm dynamics when the arm is moving. This is reflected in the torque and tension measurements again - localized to when the arm is in motion ($t = 0s$ to $2s$, $5s$ to $7s$, *etc.*).

Overall, we see that changing these parameters induce explainable and

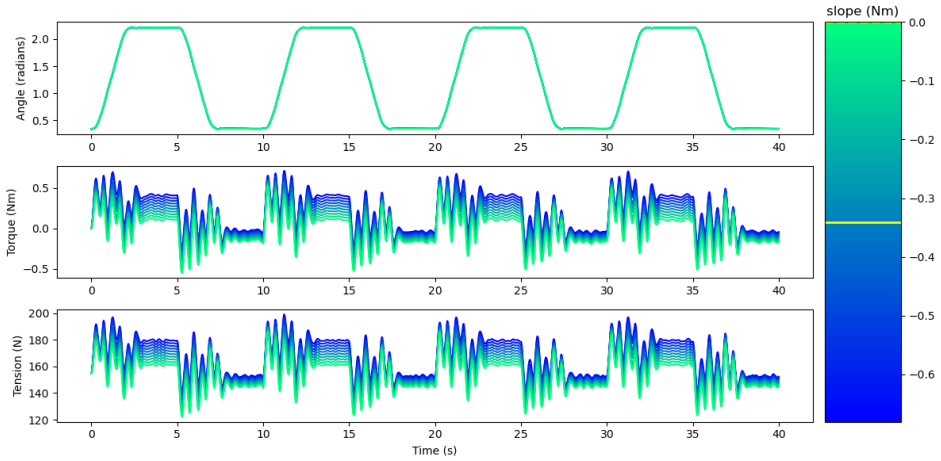


Figure 4.6: Changing the slope parameter ($slope_1$) in the PID controlled robot arm simulator.

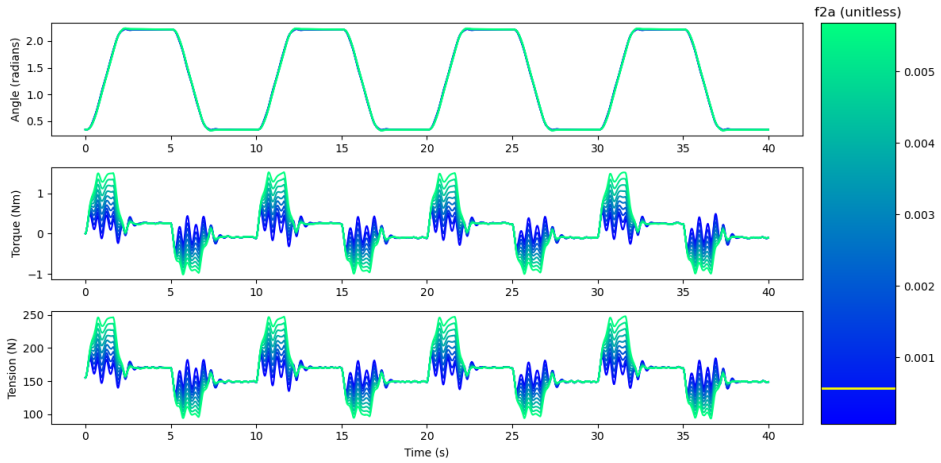


Figure 4.7: Changing the arm pulley viscous friction (f_{2a}) in the PID controlled robot arm simulator.

reasonable effects in the robot arm simulator, and extrapolate that this is a good model of the robot arm for the purposes of generating reasonable failure data. The goal is then to use the simulator to compare a multivariate OD algorithm with GVFOD and the online UDE.

Prior to starting the experiment, we need to have a sense of what non-stationary normal machine operation looks like. Using our previous definitions of fault and failure - where fault is an event that precedes incipient failure - we can define the [idealized] models of machine health shown in Figure 4.8. The machine health presented here is inversely correlated to the outlier score.

When the machine's normal operation is stationary, the machine health will stay consistent until a fault occurs. This would be best-case scenario for CBM, since the distribution of normal data encountered in training is the same as that in testing. However, we saw that this was not the case on the robot-arm dataset. Instead, if we assume that normal data is accurately modelled as gradual linear degradation, we define fault to be the event where there is a change in the degradation rate, and health profile becomes non-linear. This acceleration in degradation will predict the upcoming failure. In this situation, there is a lead time between the occurrence of machine fault and the occurrence of machine failure. A limitation of this definition is that it is perfectly plausible that failures can occur with no fault (such as in electronic systems, or in systems that wear only a very small amount and then fail suddenly), but we do not consider them here.

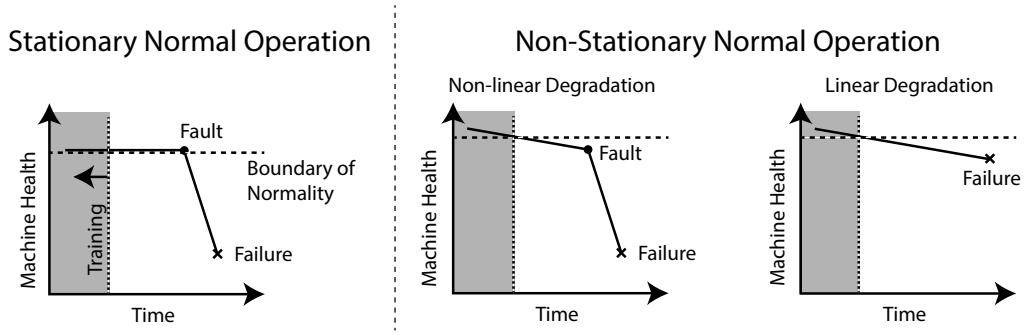


Figure 4.8: Our model of non-stationary machine operation, fault, and failure.

In Figure 4.9, we see the simulator being used to a create non-stationary normal data, and the data following the occurrence of a fault. The belt is gradually loosening, characteristic of a belt aging in application [18]. As the static belt tension (T_{base}) drops below $145N$, a fault occurs, and the rate of degradation increases. This figure exists to visualize how we generate data over a very short time scale (only three period), and a full experiment over a longer period of time will be done.

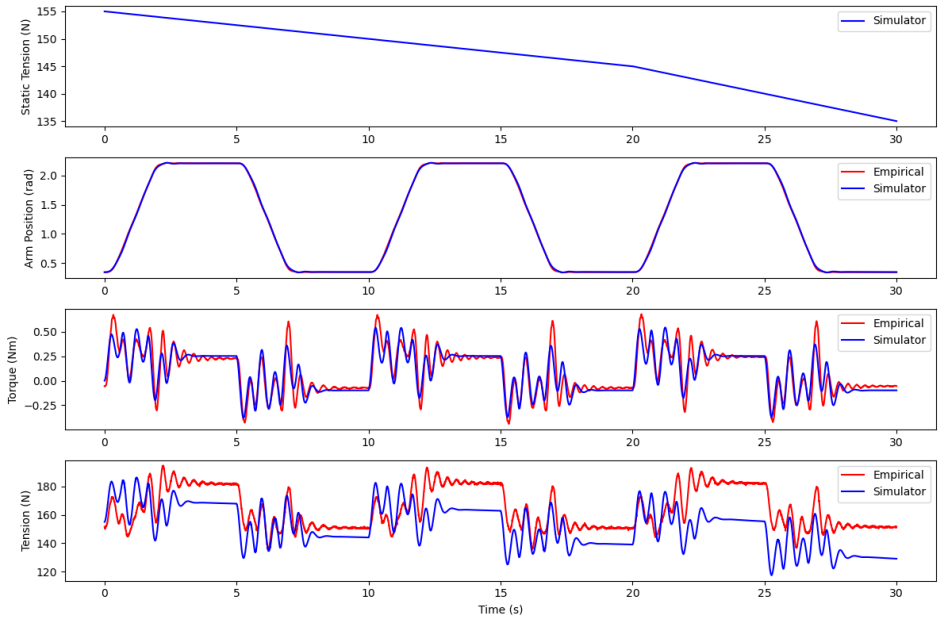


Figure 4.9: Simulated controlled robot arm data with gradual degradation and fault.

4.3 Results and Discussion

In the previous section we introduced the non-stationary environment of fault detection, with 2 periods of normal machine operation, and a single period of operation after the fault occurs. In Figure 4.10, we see the difference between LOF (a multivariate OD algorithm), GVFOD, and UDE when presented with the gradual failure data. For LOF and GVFOD, the training phase is from $0s$ to $4000s$. Afterwards (from $4000s$ to $15000s$), these algorithms are used to calculate the outlier scores of the test data. The difference between GVFOD and our representative multivariate OD algorithm is clear - GVFOD has the lowest outlier scores towards the end of training (save for some noise in the beginning), whereas the LOF scores are consistent for the entirety of the training data. Furthermore, the time at which test outlier scores rises above the outlier threshold is different - LOF starts detecting faulty behaviour soon after the end of training, at around $5500s$. Whereas, the GVFOD algorithm starts detecting faulty data much later, at around $8000s$. Since the fault occurs at $10000s$, LOF misclassifies significantly more normal data than GVFOD.

This is simply explained by knowing that LOF is learning a model of normality for the entire set of normal data. Whereas for GVFOD, because the GVFs have learned models best adapted to the end of training, this also generalizes well to the system dynamics at the start of testing. Another way of viewing this is, since GVFOD has learned a model tuned for the end of training, it is more tolerant towards the amount of model drift that occurred since the beginning of training to the end of training, and thus is more tolerant of model drift from the beginning of testing.

Why does the online UDE differ from GVFOD? In GVFOD, the model is kept fixed, *i.e.* the GVF parameters at the end of training (in this case, at $4000s$), are used to calculate the outlier score for all other observations. In UDE, learning happens continually, in an online and incremental manner, so the model parameters are changing with each time step. With the right step-size, we are able to have a consistent UDE value up until the fault occurs.[2]. In doing so, online UDE bests GVFOD and LOF in identifying the fault, showing a distinct elbow just past $10000s$. This elbow is much closer to closer to the true fault occurrence at $10,000s$, than the times at which LOF and GVFOD scores crossed their outlier threshold. Learning the proper threshold for UDE in order to classify samples as normal or abnormal, as well as learning the step-size parameter, is a potential topic of future study. We can extrapolate from the results in Figure 4.10 that UDE has the potential to be a better fault detection algorithm than GVFOD and LOF in online fault-detection settings.

There needs to be some reconciliation between the results in this chapter and those in Chapter 3. First, we have defined fault in two different ways - in Chapter 3, fault data was a set of sequential time series where the machine was physically modified to collect data simulating a near-failure condition. In this chapter, we have defined fault as an event that causes a change in rate of non-stationarity/degradation of the machine. These two potentially conflicting definitions arises from the definition of the problem - Chapter 3 framed fault detection as a semi-supervised learning task, and this chapter frames it as an

[2]Matching this step-size to the rate of non-stationarity is done empirically - in Figure 4.10, the TD learning parameter α was tuned by hand.

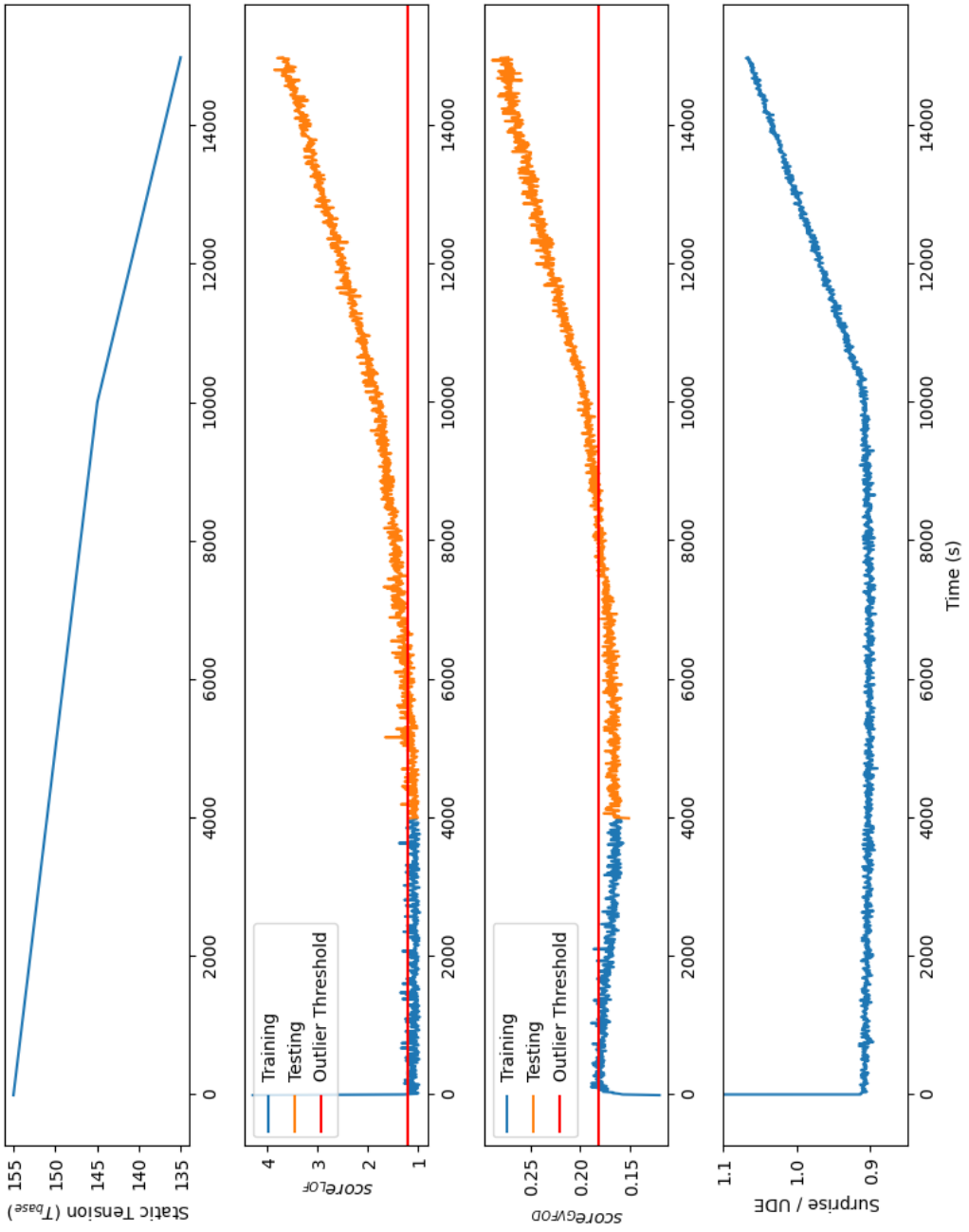


Figure 4.10: Outlier scores of LOF, GVFOD, as well as UDE on gradual failure data. The data is collected from the controlled robot arm simulator, with gradual degradation, and a fault occuring at 10000s.

online / continuous learning task. We leave the reader to decide which setting best fits their application, and choose the best algorithm accordingly.

4.4 Advantages and Limitations of Simulation

The two primary advantages of using the simulation are consistency and the ability to be modified. If parameters are kept consistent, time series data generated by the simulator will be consistent over time (stationary). Secondly, the simulated machine can be modified while it is running. This is not possible on the physical robot.

There were a few limitations to the simulator. First, the optimization was done only over the parameters that affected the arm's dynamics, but not the control dynamics. Instead, we optimized the control dynamics by hand, and they only roughly match (as seen in Figure 4.4). This optimization could have been done numerically, but we chose not to since there was no benefit for our application. Furthermore, we only validated the arm dynamics with four more periods post-training. In order to more confidently believe in the results of the simulator, more validation could be done, perhaps using fault data. This could determine how much the dynamics can be generalized to other operating conditions.

4.5 Conclusion

In this chapter, we presented the robot arm simulator first developed by Lipsett *et al.* [25], and modified it to emulate the behaviour of the robot arm device. We implemented a PID controller so that the simulator can output position, torque, and tension data indefinitely, and with a number of explainable physical parameters. By modifying these parameters, we simulated non-stationary normal operation, induced a fault event where the rate of degradation increased, and used this data to compare the performance of a multivariate OD algorithm, GVFOD, and online UDE. It was found that because GVFOD models not only normal behaviour, but also the amount of non-stationarity that exists in normal behaviour, it is better suited for generalizing to future non-

stationary machine behaviour. Overall, it was found that online UDE, with an appropriate step-size, is best suited for finding fault events. We hope that these findings allow future users of these algorithms to have a deeper understanding of the challenges of CBM in non-stationary environments, and hope that future advancements in RL and GVF research can likewise be adapted to commercial machine monitoring and maintenance applications. In doing so, industrial processes could become more reliable, safer, and more economical as they leverage machine intelligence towards the production of goods.

Chapter 5

Conclusion

We conclude this thesis by reintroducing the initial thesis statement posed.

Can we use general value functions and temporal difference learning with native data to detect faults and assess machine health in machines?

We first looked at the offline setting of machine fault detection, where we used condition monitoring (CM) data collected from a robot-arm machine operating normally, and aimed to identify future test data as either normal or faulty. We introduced the GVFOD algorithm, which uses a group of general value functions to form a predictive model of the machine using the CM data. Test data is mapped to an outlier score using unexpected demon error, indicating whether the machine is in good condition or not. When comparing different outlier detection algorithms with General Value Function Outlier Detection (GVFOD), it was found that all methods are equally capable of detecting faulty behaviour; whereas certain algorithms make more errors identifying normal behaviour than others. In general, it was found that GVFOD outperforms other algorithms when presented with sufficient quantities (over 4 hours) of training data, albeit with higher variance. Moreover, with a background in knowledge and the machine dynamics, GVFOD was shown to outperform outlier detection methods, when no fault data is available for parameter selection. GVFOD was the only method to consistently perform better with the addition of more training data, whereas some other methods would perform worse. This outlines the importance of accounting for non-stationarity in normal machine

operation. To the best of our knowledge, GVFOD is the first application of general value functions in an offline fault detection setting. This RL-based outlier detection method promises to improve user-belief in a fault detection system by reporting fewer false positives, and significantly advances the feasibility of data-driven condition based maintenance systems.

We then created an online setting of machine fault detection using a simulator based upon the robot-arm apparatus. Using this, we were able to model a machine operating a normal situation with gradual degradation. A fault was induced, which caused an accelerated rate of wear. A representative multivariate outlier detection algorithm, local outlier factor (LOF), was compared to GVFOD and the online unexpected demon error (UDE) metric. From this, we were able to better understand the differences between LOF and GVFOD - LOF makes a model of normal machine operation; but GVFOD also models the amount of wear that is considered normal. However, comparing both to UDE demonstrated that the online UDE algorithm has potential to be better than both LOF and GVFOD in online fault detection, as it better measure the rate of wear in the machine. Nonetheless, this online setting of fault detection led us to better understand the promising performance of GVFOD in offline settings.

This thesis is yet another demonstration of the applicability of reinforcement learning methods towards industrial processes; in this case, to ensuring process reliability with CBM. We hope that the findings can be implemented in both the industrial production of goods, and potentially in consumer and household goods as well, so machines can be safer, more reliable, and more economical in the future.

References

[1] M. Aoki, *State Space Modeling of Time Series*, en, ser. Universitext. Berlin Heidelberg: Springer-Verlag, 1987, ISBN: 978-3-642-96985-0. DOI: 10.1007/978-3-642-96985-0. [Online]. Available: https://www.springer.com/gp/book/9783642969850 (visited on 10/28/2020).

[2] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization," in *Advances in Neural Information Processing Systems 24*, J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2011, pp. 2546–2554. [Online]. Available: http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf (visited on 06/22/2020).

[3] J. Bergstra, D. Yamins, and D. Cox, "Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms," en, Austin, Texas, 2013, pp. 13–19. DOI: 10.25080/Majora-8b375195-003. [Online]. Available: https://conference.scipy.org/proceedings/scipy2013/bergstra_hyperopt.html (visited on 07/06/2020).

[4] M. Breunig, H.-P. Kriegel, R. Ng, and J. Sander, "LOF: Identifying Density-Based Local Outliers.," vol. 29, Jun. 2000, pp. 93–104. DOI: 10.1145/342009.335388.

[5] *Calculating the Cost of Downtime*, en-GB, Mar. 2019. [Online]. Available: https://www.spartansolutions.com/blog/calculating-the-cost-of-downtime/ (visited on 11/10/2020).

[6] W. D. Callister and D. G. Rethwisch, *Materials Science and Engineering An Introduction*, 8th. John Wiley & Sons, Ltd, 2010.

[7] A. Carreño, I. Inza, and J. A. Lozano, "Analyzing rare event, anomaly, novelty and outlier detection terms under the supervised classification framework," en, *Artificial Intelligence Review*, vol. 53, no. 5, pp. 3575–3594, Jun. 2020, ISSN: 1573-7462. DOI: 10.1007/s10462-019-09771-y. [Online]. Available: https://doi.org/10.1007/s10462-019-09771-y (visited on 10/07/2020).

[8] V. Chandola, A. Banerjee, and V. Kumar, "Outlier detection: A survey," *ACM Computing Surveys*, vol. 14, p. 15, 2007.

[9] ——, "Anomaly detection: A survey," en, *ACM Computing Surveys*, vol. 41, no. 3, pp. 1–58, Jul. 2009, ISSN: 03600300. DOI: 10.1145/1541880.1541882. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1541880.1541882 (visited on 02/18/2020).

[10] M. Christ, N. Braun, J. Neuffer, and A. W. Kempa-Liehr, "Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package)," en, *Neurocomputing*, vol. 307, pp. 72–77, Sep. 2018, ISSN: 0925-2312. DOI: 10.1016/j.neucom.2018.03.067. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231218304843 (visited on 06/23/2020).

[11] C. Cortes and V. Vapnik, "Support-vector networks," en, *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995, ISSN: 0885-6125, 1573-0565. DOI: 10.1007/BF00994018. [Online]. Available: http://link.springer.com/10.1007/BF00994018 (visited on 10/26/2020).

[12] A. Foss, O. R. Zaïane, and S. Zilles, "Unsupervised Class Separation of Multivariate Data through Cumulative Variance-Based Ranking," en, in *2009 Ninth IEEE International Conference on Data Mining*, tex.ids: foss_unsupervised_2009 ISSN: 2374-8486, Miami Beach, FL, USA: IEEE, Dec. 2009, pp. 139–148, ISBN: 978-1-4244-5242-2. DOI: 10.1109/ICDM.2009.17. [Online]. Available: http://ieeexplore.ieee.org/document/5360239/ (visited on 01/01/2020).

[13] M. Goldstein and A. Dengel, "Histogram-based Outlier Score (HBOS): A fast Unsupervised Anomaly Detection Algorithm," Sep. 2012.

[14] E. Graves, *Tile Coder*, en, 2020. [Online]. Available: https://github.com/gravesec/actor-critic-with-emphatic-weightings (visited on 11/02/2020).

[15] J. Günther, A. Kearney, M. Dawson, C. Sherstan, and P. Pilarski, "Predictions, Surprise, and Predictions of Surprise in General Value Function Architectures," Oct. 2018.

[16] J. Günther, P. M. Pilarski, G. Helfrich, H. Shen, and K. Diepold, "Intelligent laser welding through representation, prediction, and control learning: An architecture with deep neural networks and reinforcement learning," en, *Mechatronics*, vol. 34, pp. 1–11, Mar. 2016, ISSN: 09574158. DOI: 10.1016/j.mechatronics.2015.09.004. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0957415815001555 (visited on 11/16/2020).

[17] M. Gupta, Jing Gao, C. C. Aggarwal, and Jiawei Han, "Outlier Detection for Temporal Data: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2250–2267, 2014, ISSN: 1041-4347. DOI: 10.1109/TKDE.2013.184. [Online]. Available: https://ieeexplore.ieee.org/document/6684530.

[18] S. S. Hamza, "Effect of aging and carbon black on the mechanical properties of EPDM rubber," en, *Polymer Testing*, vol. 17, no. 2, pp. 131–137, Apr. 1998, ISSN: 0142-9418. DOI: 10.1016/S0142-9418(97)00039-1. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0142941897000391 (visited on 11/12/2020).

[19] D. M. Hawkins, *Identification of Outliers*, en. Dordrecht: Springer Netherlands, 1980, ISBN: 978-94-015-3996-8 978-94-015-3994-4. DOI: 10.1007/978-94-015-3994-4. [Online]. Available: http://link.springer.com/10.1007/978-94-015-3994-4 (visited on 11/05/2020).

[20] V. J. Hodge and J. Austin, "A Survey of Outlier Detection Methodologies," en, *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 2004.

[21] A. K. S. Jardine, D. Lin, and D. Banjevic, "A review on machinery diagnostics and prognostics implementing condition-based maintenance," en, *Mechanical Systems and Signal Processing*, vol. 20, no. 7, pp. 1483–1510, Oct. 2006, ISSN: 0888-3270. DOI: 10.1016/j.ymssp.2005.09.012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0888327005001512 (visited on 11/25/2019).

[22] M. Kay, S. N. Patel, and J. A. Kientz, "How Good is 85%?: A Survey Tool to Connect Classifier Evaluation to Acceptability of Accuracy," en, in *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems - CHI '15*, Seoul, Republic of Korea: ACM Press, 2015, pp. 347–356, ISBN: 978-1-4503-3145-6. DOI: 10.1145/2702123.2702603. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2702123.2702603 (visited on 12/29/2019).

[23] E. Keogh, J. Lin, S.-H. Lee, and H. V. Herle, "Finding the most unusual time series subsequence: Algorithms and applications," en, *Knowledge and Information Systems*, vol. 11, no. 1, pp. 1–27, Jan. 2007, ISSN: 0219-3116. DOI: 10.1007/s10115-006-0034-6. [Online]. Available: https://doi.org/10.1007/s10115-006-0034-6 (visited on 10/24/2020).

[24] H.-P. Kriegel, M. S hubert, and A. Zimek, "Angle-based outlier detection in high-dimensional data," en, in *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 08*, Las Vegas, Nevada, USA: ACM Press, 2008, p. 444, ISBN: 978-1-60558-193-4. DOI: 10.1145/1401890.1401946. [Online]. Available: http://dl.acm.org/citation.cfm?doid=1401890.1401946 (visited on 07/04/2020).

[25] M. Lipsett, A. Maltais, M. Riazi, N. Olmedo, and O. Zaiane, "Robot Manipulator Drive Fault Diagnostics Using Data-Driven and Analytical Modelling," p. 12, May 2019.

[26] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation Forest," en, in *2008 Eighth IEEE International Conference on Data Mining*, Pisa, Italy: IEEE, Dec. 2008, pp. 413–422, ISBN: 978-0-7695-3502-9. DOI: 10.1109/ICDM.2008.17. [Online]. Available: http://ieeexplore.ieee.org/document/4781136/ (visited on 07/03/2020).

[27] M. Markou and S. Singh, "Novelty detection: A review—part 1: Statistical approaches," en, *Signal Processing*, vol. 83, no. 12, pp. 2481–2497, Dec. 2003, ISSN: 0165-1684. DOI: 10.1016/j.sigpro.2003.07.018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165168403002020 (visited on 10/18/2020).

[28] ——, "Novelty detection: A review—part 2:: Neural network based approaches," en, *Signal Processing*, vol. 83, no. 12, pp. 2499–2521, Dec. 2003, ISSN: 0165-1684. DOI: 10.1016/j.sigpro.2003.07.019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165168403002032 (visited on 10/18/2020).

[29] J. Modayil, A. White, and R. S. Sutton, "Multi-timescale nexting in a reinforcement learning robot," *Adaptive Behavior*, vol. 22, no. 2, pp. 146–160, 2014, ISSN: 1059-7123. DOI: 10.1177/1059712313511648. [Online]. Available: https://journals.sagepub.com/doi/full/10.1177/1059712313511648.

[30] M. Pecht, "Prognostics and Health Management of Electronics," en, in *Encyclopedia of Structural Health Monitoring*, American Cancer Society, 2009, ISBN: 978-0-470-06162-6. DOI: 10.1002/9780470061626.shm118. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470061626.shm118 (visited on 02/18/2020).

[31] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," en, *Signal Processing*, vol. 99, pp. 215–249, Jun. 2014, ISSN: 01651684. DOI: 10.1016/j.sigpro.2013.12.026. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S016516841300515X (visited on 10/12/2020).

[32] A. Prisacaru, P. J. Gromala, M. B. Jeronimo, B. Han, and G. Q. Zhang, "Prognostics and health monitoring of electronic system: A review," in *2017 18th International Conference on Thermal, Mechanical and Multi-Physics Simulation and Experiments in Microelectronics and Microsystems (EuroSimE)*, Apr. 2017, pp. 1–11. DOI: 10.1109/EuroSimE.2017.7926248.

[33] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient Algorithms for Mining Outliers from Large Data Sets," en, *ACM SIGMOD Record*, vol. 29, pp. 427–438, 2000.

[34] P. G. Ranky, "Automotive robotics," *Industrial Robot: An International Journal*, vol. 31, no. 3, pp. 252–257, Jan. 2004, Publisher: Emerald Group Publishing Limited, ISSN: 0143-991X. DOI: 10.1108/01439910410532314. [Online]. Available: https://doi.org/10.1108/01439910410532314 (visited on 11/10/2020).

[35] M. Riazi, "Detecting the Onset of Machine Failure Using Anomaly Detection Methods," PhD thesis, University of Alberta, 2019.

[36] M. Riazi, O. Zaiane, T. Takeuchi, A. Maltais, J. Günther, and M. Lipsett, "Detecting the Onset of Machine Failure Using Anomaly Detection Methods," en, in *Big Data Analytics and Knowledge Discovery*, C. Ordonez, I.-Y. Song, G. Anderst-Kotsis, A. M. Tjoa, and I. Khalil, Eds., Series Title: Lecture Notes in Computer Science, vol. 11708, Cham: Springer International Publishing, 2019, pp. 3–12, ISBN: 978-3-030-27519-8 978-3-030-27520-4. DOI: 10.1007/978-3-030-27520-4_1. [Online]. Available: http://link.springer.com/10.1007/978-3-030-27520-4_1 (visited on 05/01/2020).

[37] P. J. Rousseeuw, "Least Median of Squares Regression," *Journal of the American Statistical Association*, vol. 79, no. 388, pp. 871–880, Dec. 1984, ISSN: 0162-1459. DOI: 10.1080/01621459.1984.10477105. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/01621459.1984.10477105 (visited on 10/23/2020).

[38] P. Rousseeuw and K. Driessen, "A Fast Algorithm for the Minimum Covariance Determinant Estimator," *Technometrics*, vol. 41, pp. 212–223, Aug. 1999. DOI: 10.1080/00401706.1999.10485670.

[39] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt, "Support Vector Method for Novelty Detection," in *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen, and K. Müller, Eds., MIT Press, 2000, pp. 582–588. [Online]. Available: http://papers.nips.cc/paper/1723-support-vector-method-for-novelty-detection.pdf (visited on 07/04/2020).

[40] *Scipy.integrate.solve_ivp — SciPy v1.5.1 Reference Guide*. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html#id11 (visited on 07/16/2020).

[41] R. S. Sutton, *Tile Coding Software – Reference Manual, Version 3 beta*. [Online]. Available: http://incompleteideas.net/tiles/tiles3.html (visited on 12/31/2019).

[42] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, en, Second edition, ser. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018, ISBN: 978-0-262-03924-6.

[43] R. S. Sutton, J. Modayil, M. Delp, T. Degris, P. M. Pilarski, A. White, and D. Precup, "Horde: A Scalable Real-time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction," en, in *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems*, Tumer, Yolum, Sonenberg, and Stone, Eds., Taipei, Taiwan: International Foundation for Autonomous Agents and Multiagent Systems, May 2011, pp. 761–768.

[44] R. S. Sutton and B. Tanner, "Temporal-Difference Networks," in *Advances in Neural Information Processing Systems 17*, L. K. Saul, Y. Weiss, and L. Bottou, Eds., MIT Press, 2005, pp. 1377–1384. [Online]. Available: http://papers.nips.cc/paper/2545-temporal-difference-networks.pdf (visited on 10/31/2020).

[45] *Time constant*, en, Page Version ID: 973815119, Aug. 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Time_constant&oldid=973815119 (visited on 11/12/2020).

[46] R. S. Tsay, "Time Series and Forecasting: Brief History and Future Research," *Journal of the American Statistical Association*, vol. 95, no. 450, pp. 638–643, 2000, Publisher: [American Statistical Association, Taylor & Francis, Ltd.], ISSN: 0162-1459. DOI: 10.2307/2669408. [Online]. Available: http://www.jstor.org/stable/2669408 (visited on 10/27/2020).

[47] N. Vichare and M. Pecht, "Prognostics and health management of electronics," *IEEE Transactions on Components and Packaging Technologies*, vol. 29, no. 1, pp. 222–229, Mar. 2006, ISSN: 1557-9972. DOI: 10.1109/TCAPT.2006.870387.

[48] A. White, "Developing a predictive approach to knowledge," en, PhD thesis, University of Alberta, 2015.

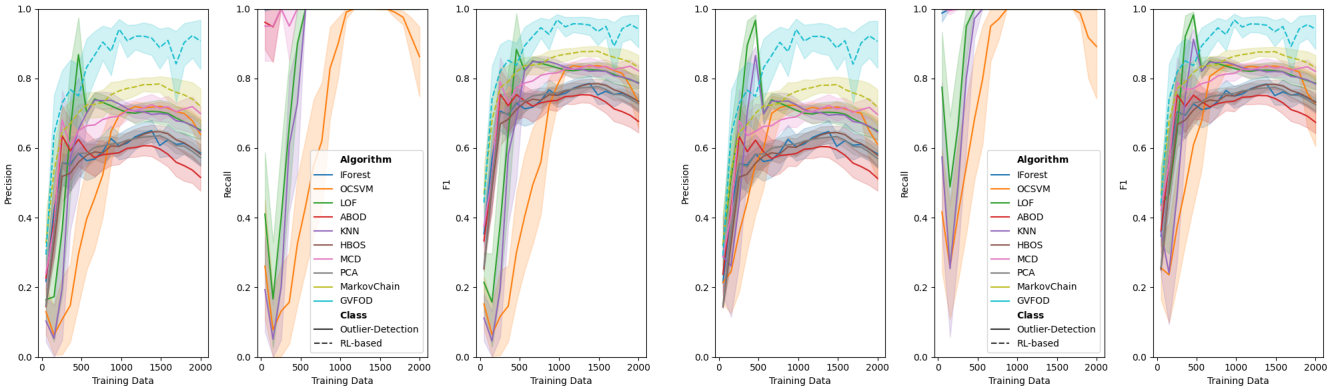
[49] A. White, J. Modayil, and R. S. Sutton, "Surprise and Curiosity for Big Data Robotics," en, in *Sequential Decision-Making with Big Data: Papers from the AAAI-14 Workshop*, 2014, p. 5.

[50] Y. Zhao, Z. Nasrullah, and Z. Li, "PyOD: A Python Toolbox for Scalable Outlier Detection," no. Journal Article, 2019. [Online]. Available: https://arxiv.org/abs/1901.01588.

Appendix A

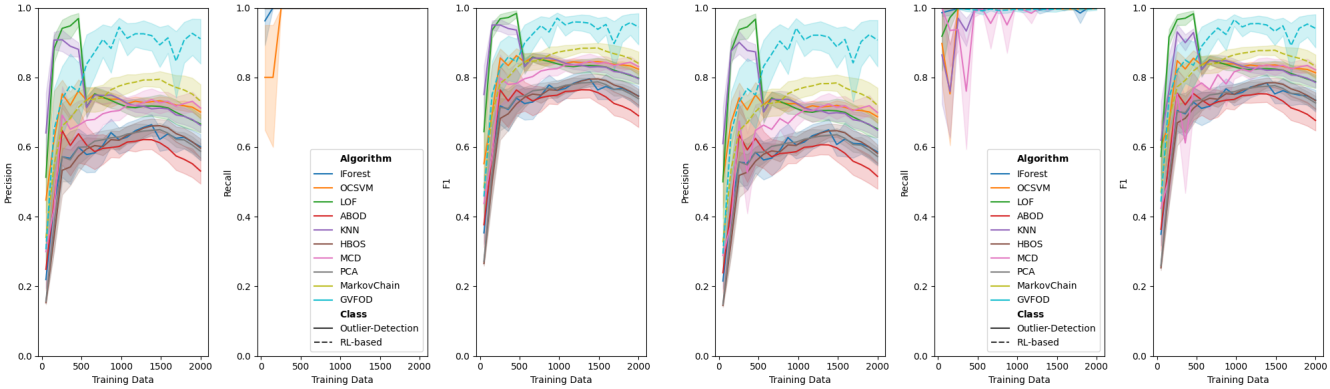
Extended Results for GVFOD

The following figures accompany those found in Chapter 3, where only the *Loose L1* class of failure was reported. It can be seen that no substantive difference in performance between fault classes is seen, and that the analysis in Chapter 3 generalizes to all five failure modes.



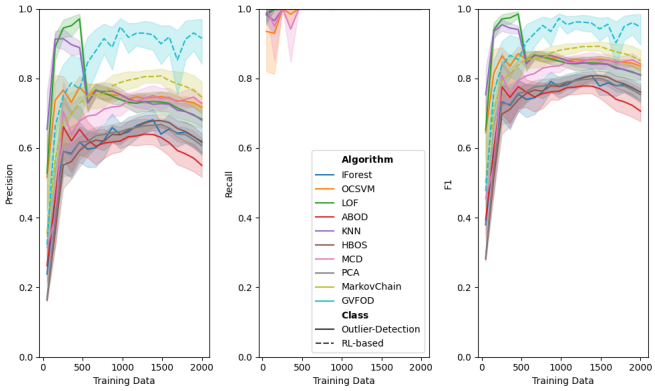
(a) *Loose L1*

(b) *Loose L2*



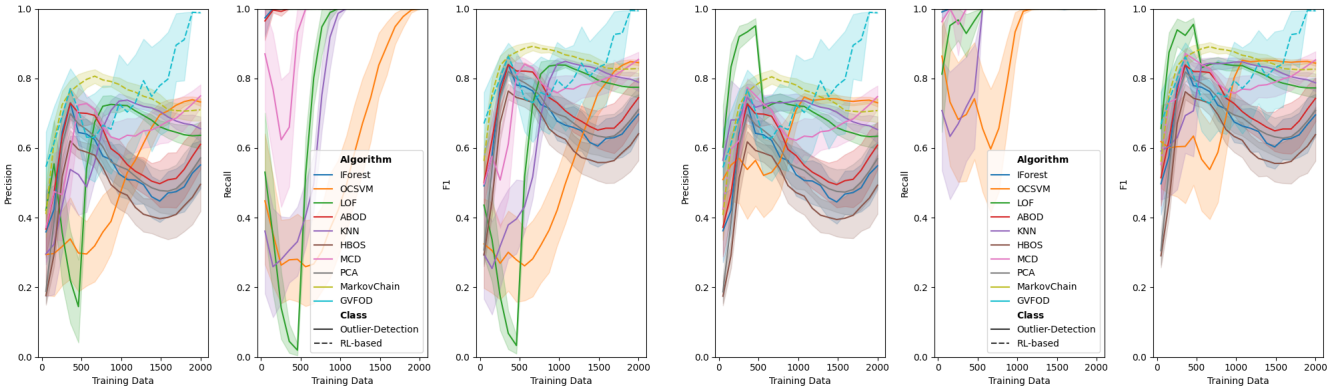
(c) *Tight*

(d) *Sandy*



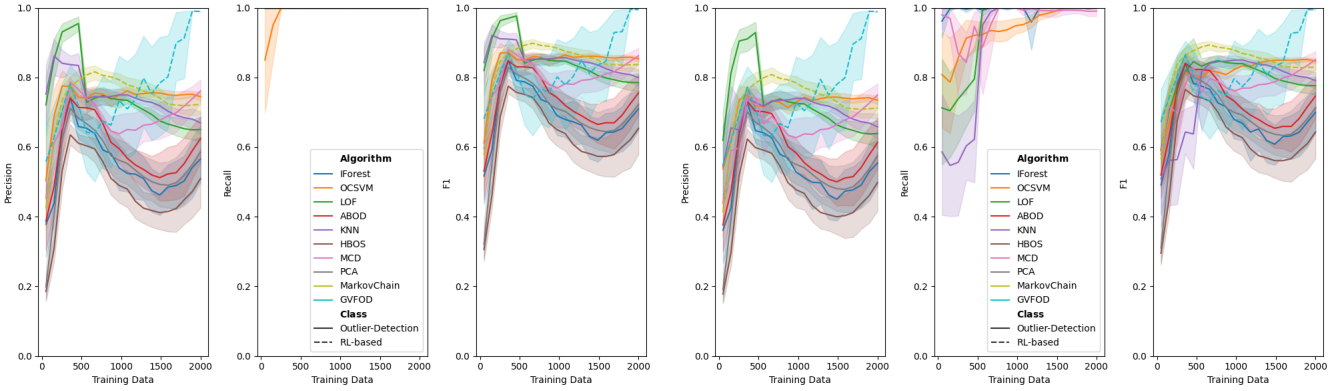
(e) *High Temperature*

Figure A.1: Algorithm comparison on *hyperparameter tuning* data with optimized parameters. No delay.



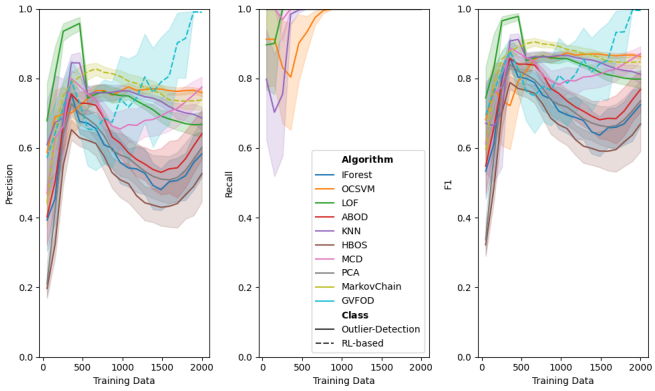
(a) *Loose L1*

(b) *Loose L2*



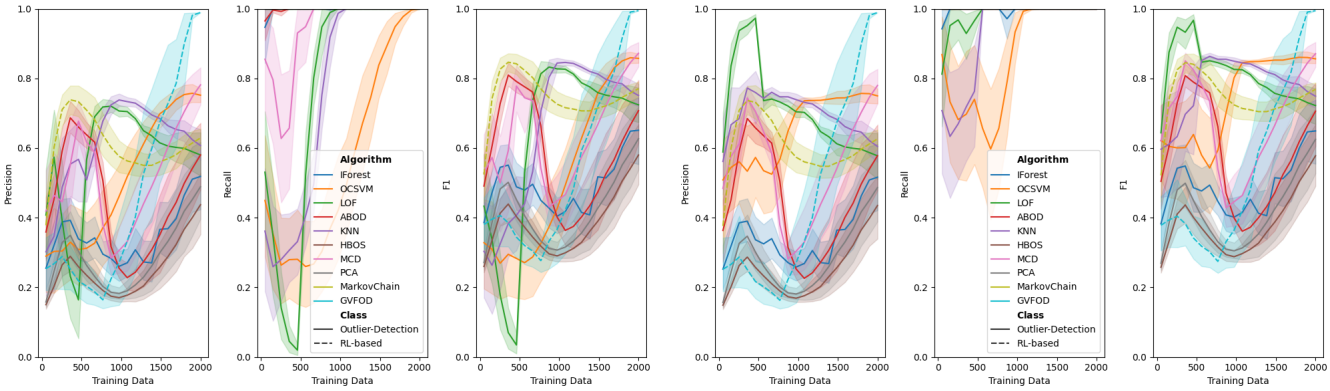
(c) *Tight*

(d) *Sandy*



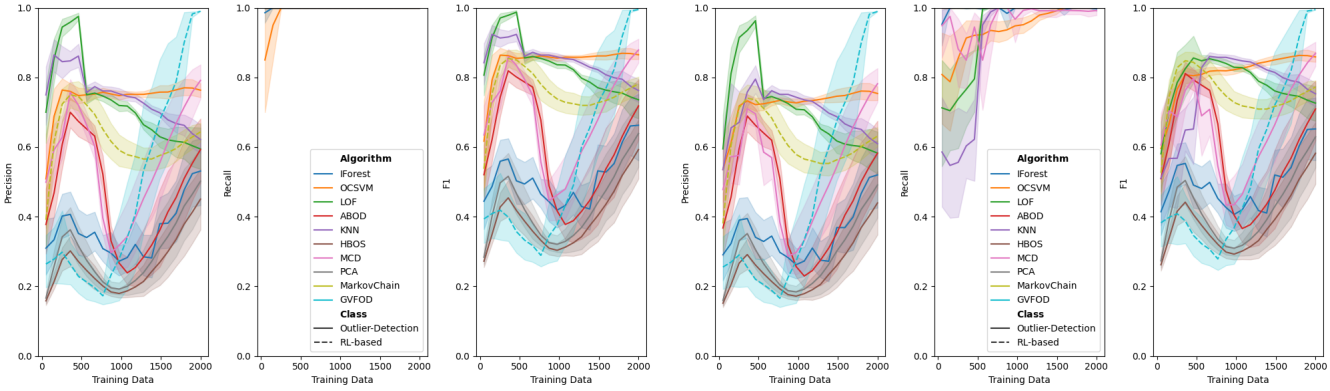
(e) *High Temperature*

Figure A.2: Algorithm comparison on *validation* data with optimized parameters. No delay.



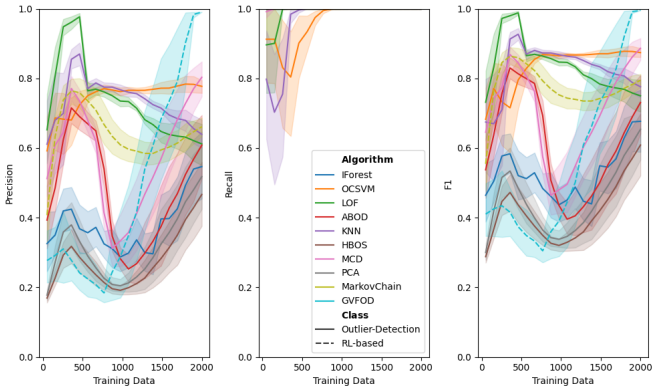
(a) *Loose L1*

(b) *Loose L2*



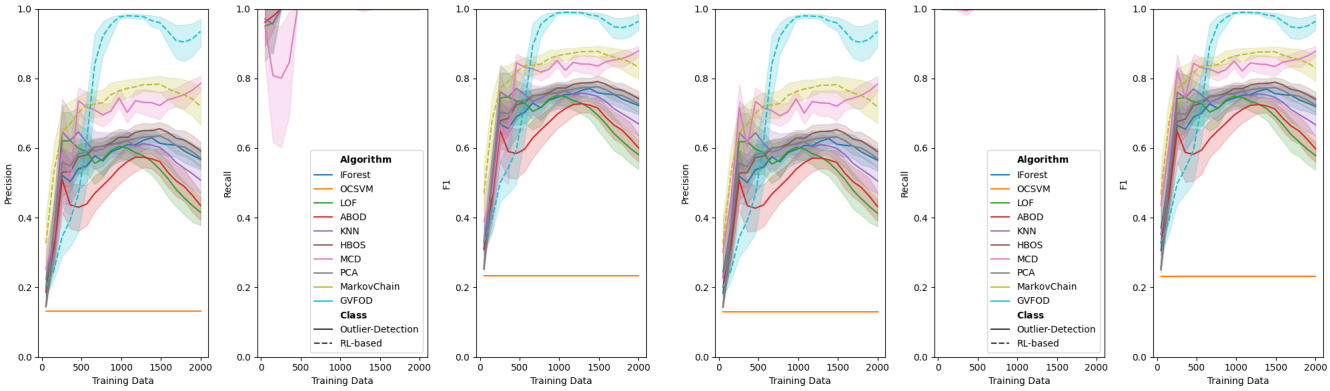
(c) *Tight*

(d) *Sandy*



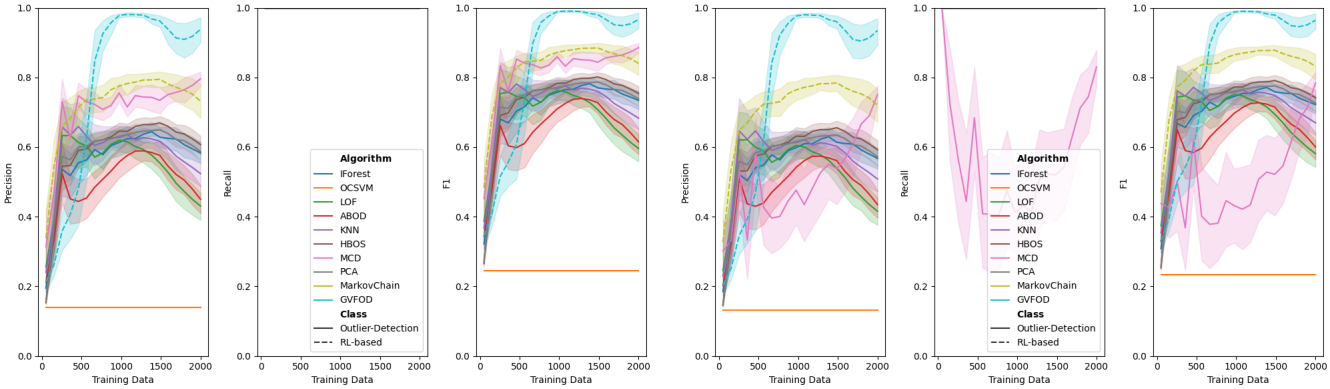
(e) *High Temperature*

Figure A.3: Algorithm comparison on *validation* data with optimized parameters. 2 hour delay.



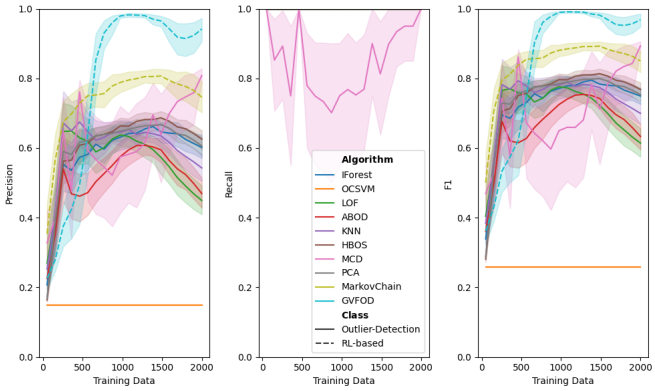
(a) *Loose L1*

(b) *Loose L2*



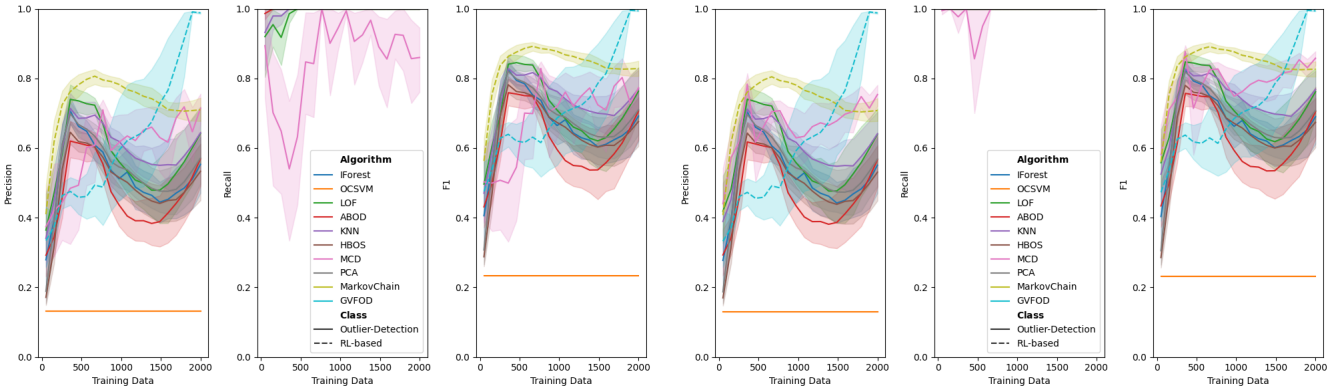
(c) *Tight*

(d) *Sandy*



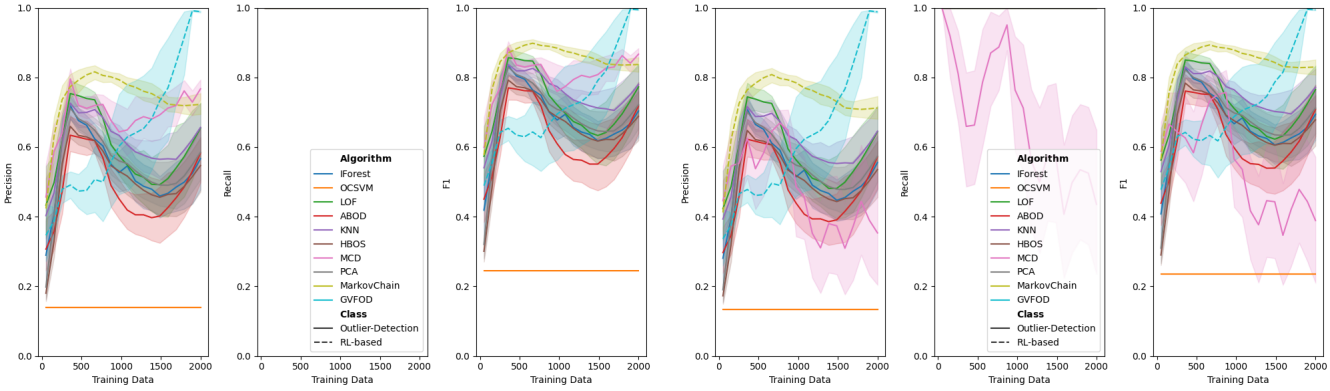
(e) *High Temperature*

Figure A.4: Algorithm comparison on *hyperparameter tuning* data with default parameters. No delay.



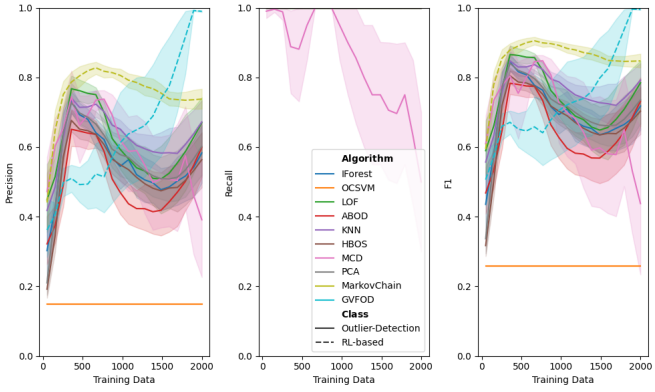
(a) *Loose L1*

(b) *Loose L2*



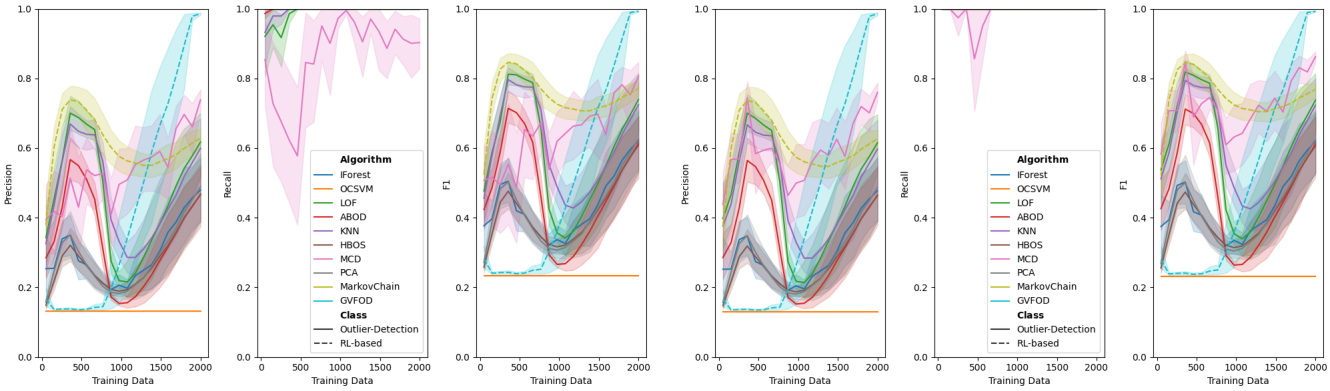
(c) *Tight*

(d) *Sandy*



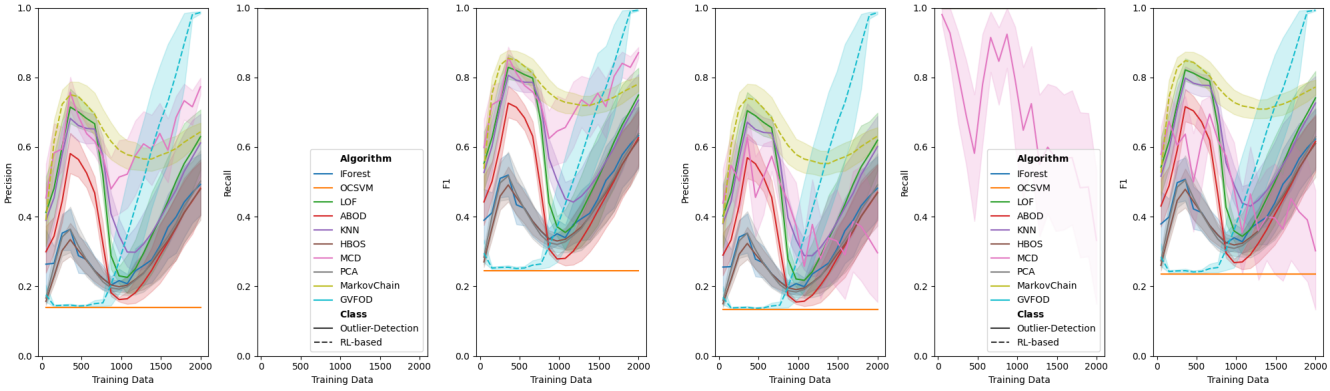
(e) *High Temperature*

Figure A.5: Algorithm comparison on *validation* data with default parameters. No delay.



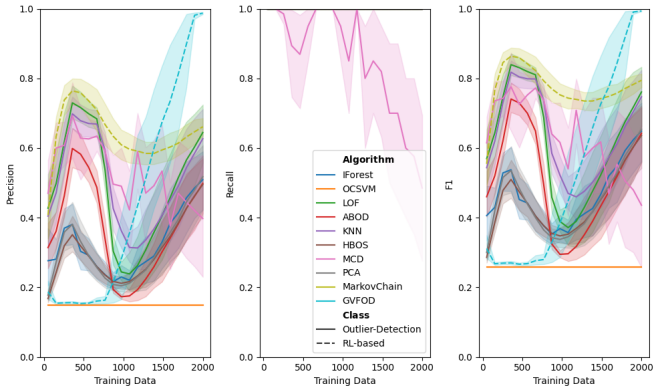
(a) *Loose L1*

(b) *Loose L2*



(c) *Tight*

(d) *Sandy*



(e) *High Temperature*

Figure A.6: Algorithm comparison on *validation* data with default parameters. 2 hour delay.