University of Technology Sydney
Faculty of Engineering and Information Technology

Reliability of Artificial Intelligence

Capstone Report

Ajal Singh

Student Number: 12621189
Project Number: AUT-21-04015
Major: Mechatronics
Supervisor: Dr Diep Nguyen

May 31, 2021

Statement of Originality

I, Ajal Singh, declare that I am the sole author of this report entitled, *Reliability of Artificial Intelligence*. I have not used fragments of text from other sources without proper acknowledgement/reference. All theories, results and designs incorporated into this report that do not belong to me have been appropriately referenced and all sources of assistance have been acknowledged.

Signed: Ajal.S, May 31, 2021

Abstract

Reliability of Artificial Intelligence

Ajal Singh
Autumn 2021

Artificial Intelligence and specifically Machine Learning is becoming more prominent throughout various applications. Thanks to recent innovations in IoT and Big Data, AI/ML systems are becoming increasingly complex in high-risk scenarios such as smart cities, healthcare and cybersecurity. However, AI/Ml has been proven as troublesome in numerous applications and calls into question the reliability of AI.

Machine Leaning is a broad area of interest and has multiple branches, including supervised, unsupervised and reinforcement learning. Each branch has many factors which could affect the reliability of a model.

This report is focused on the development and impacts of bias within datasets and the intended deployment environment in supervised learning models. Also discussed is the issue of model misspecification, where inappropriate algorithms/methods are utilised hence preventing robustness and failing to meet intended outcomes. Finally, reward hacking is a problematic phenomenon native to Reinforcement Learning systems. Here the agent exploits its reward function to achieve maximum reward without accomplishing the intended task. This report aims to not only determine how aforementioned reliability issues arise in AI/ML, but also propose various recommendations on ways to mitigate them.

Acknowledgements

I wish to acknowledge my capstone supervisor, Dr Diep Nguyen, for providing his knowledge and assistance on Artificial Intelligence and Machine Learning over the past year. I also acknowledge that experimentation performed during this project was built on top of a project/dataset by Mikael Ahonen and Erno Lahtinen and is referenced accordingly throughout this report.

Contents

List of Figures

List of Tables

Nomenclature

AI	Artificial Intelligence
ML	Machine Learning
RL	Reinforcement Learning
IoT	Internet of Things
KL	Kullback-Leibler
DF	Dataframe
PDM	Predictive Maintenance
DAG	Directed Acyclic Graph
DGP	Data Generation Process
SVM	Support Vector Machine
k-NN	k-Nearest Neighbours
NN	Neural Networks
MLP	Multi-layer Perceptron
DNN	Deep Neural Networks
CNN	Convolutional Neural Networks
SGD	Stochastic Gradient Descent
CLF	Classifier
RUL	Remaining Usefule Lifetime
RMSE	Root Mean Square Error
ROC	Receiver Operating Characteristic
HOL	Hyperparameter Optimisation Machines
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negtive
MSE	Mean Square Error
MDP	Markov Decision Process
RF	Reward Function
RDP	Reward Design Problem
TMS	Traffic Management System

Engineering Research Problem

1.1 Background

As the use of Artificial Intelligence (AI) and Machine Learning (ML) continues to grow throughout the world in high-risk applications, models have become ever-increasingly complex and diverse. As a result, they often become prone to accidents where unintended and harmful behaviour is observed, and consequently are scrutinized as disruptive and unreliable solutions. The recent emergence in smart cities have seen AI and ML being used in various applications such as transportation, healthcare, environmental, and public safety as depicted in Figure (1.1).

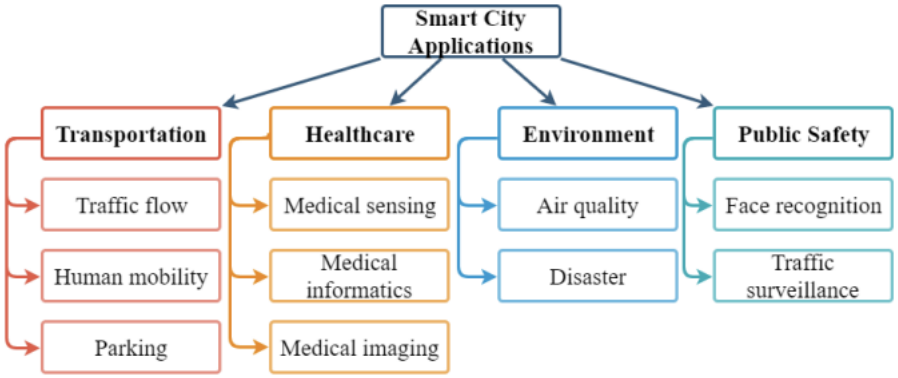


Figure 1.1: Smart City Artificial Intelligence Applications [1]

For an AI/ML system to be considered reliable, it must perform tasks when required as it was originally intended, produce consistent results using real-world data (and shifts in data), and remain robust and predictable. This means it must also fail in a predictable manner [13].

1.2 Applications

One of the most discussed and disruptive applications of ML is facial recognition systems used by authorities which fails to distinguish between darker skin individuals. This technology is used to assist the police in identifying potential criminals/suspects and often leads to wrongful arrests of dark-skinned people [14]. This example highlights the importance and the need for reliability in ML solutions.

There are many more applications where reliability is crucial due to the potential consequences. For example, cancer diagnosis systems trialled in the US fail to detect cancer in patients in differing hospitals and/or countries, resulting in death. As another example, unintended behaviours in traffic management systems would increase congestion resulting in poor ambient air quality and noise pollution.

The rapid technological changes in manufacturing have produced a boom in Industry 4.0 applications involving Artificial Intelligence, connected devices (IoT) and Big Data. A paper on use cases of AI in Industry 4.0 summarises the advantages ML, *AI with machine learning technique can automate the manufacturing process which increase the productivity, efficiency, optimize production cost and reduce manual error* [15]. A key area is predictive maintenance, where real-time equipment data is captured and historical equipment data is evaluated using AI and ML models to estimate the equipment life cycle and hence perform timely maintenance to reduce or eliminate down-time. Downtime is undesirable for manufacturers as it equates to the loss of revenue.

AI in cybersecurity helps protect enterprises by detecting unusual activity, patterns, and malicious behaviour and can respond to different situations. For manufacturers, this could be used for asset protection while banks and financial institutions may use this form ML to detect suspicious activity and fraud [15].

1.3 Project Contextualisation

A tutorial presented by Suchi Saria and Adarsh Subbaswamy of John Hopkins University [16] postulates some causes and failure prevention techniques for use in supervised learning systems (regression and classification). Some of the sources of unreliability discussed in this tutorial are inadequate data, changes in training and deployment environments, and model misspecification. These causes mentioned above will form the basis of this research project.

Another reliability issue is discussed in a separate paper, *Concrete Problems in AI Safety* [17], is the prevalence of reward hacking in Reinforcement Learning systems. *Reward hacking* is defined as the AI agents ability to cheat the system to achieve the highest reward in an unintended way. For example, a positive reward may be given to a traffic management system when there is no congestion. However, the AI model decides to divert all traffic through alternative routes, essentially shutting down this particular road/intersection. This prevents congestion but does not perform as desired. This notion is also investigated in this research project.

In Chapter 2, we outline the methodologies used throughout this report. Chapter 3 discusses how bias arises in datasets through labelling and datashift. Experimentation is performed to show that bias exists before solutions for reducing bias are described. Chapter 4 examines the factors which impact algorithm selection on a number of Artificial Intelligence applications. Subsequently, methods to improve the reliability and performance of algorithms are discussed. Afterwards, in Chapter 5, the causes and prevention approaches of reward hacking in RL systems are discussed.

1.4 Research Question

How can the reliability of Artificial Intelligence be improved against inadequate data labelling, unsuitable algorithm choices, and reward hacking?

Methodology

This project has been divided into three main sections based on the factors of unreliability mentioned earlier in Chapter 1. The main factors studied in this project are:

- Label Bias and Environmental Datashift
- Suitable Algorithm Selection
- Reward Hacking

2.1 Label Bias and Environmental Datashift

To investigate the effect of biased data labelling, we will train two models using a single independent algorithm. One will use biased data while the other uses unbiased data. These datasets will be modelled using the mathematical framework outlined in the conference paper *Identifying and Correcting Label Bias in Machine Learning* [10]. Each model will be trained and evaluated using data that has been split from the same distribution. For a model to be considered reliable, it must be able to properly generalise or adapt well to new and unseen data. A good, reliable model can achieve high accuracy (among other metrics) scores with low variance between datasets. Therefore, both of these trained models will then be fed previously unseen data (i.e. deployment data) to determine its ability to generalise.

2.2 Suitable Algorithm Selection

As shown in Figure (2.1), when it comes to AI and ML, the appropriacy of solutions or algorithms depends on elements such as the specific application and the level of supervision required. More often than not, more than one algorithm could be a viable solution (see Figure 2.2). Therefore, to investigate suitable algorithm selections, models will be trained with a single dataset using different algorithms (with different assumptions). They will then be tested for accuracy to determine suitable algorithm choices. Evaluating the reliability of a model is dependant on the model type. Accuracy, precision and recall are three common metrics we can use to evaluate a model. However, depending on particular applications, other complex means of metric evaluation may be necessary.

Machine learning algorithms	Purpose
Feed forward neural network	Smart health
Densities based clustering and regression	Smart citizen
K-means	Smart city, Smart home
Clustering & anomaly detector	Smart traffic
One class support vector machine	Smart human active control
Support vector regression	Smart whether
Linear regression	Smart market analysis

Figure 2.1: Machine Learning Algorithms for specific applications [2]

The bias-variance trade-off should be considered when optimising ML models. Bias is the model′s ability to learn the wrong things due to oversimplification or incorrect assumptions. Variance is the error due to sensitivity as a result of small fluctuations in training data. As the complexity of the model increases, bias decreases but, the variance will increase. This is the trade-off between these two factors. An overfit model is one that is too complex, resulting in high variance and low bias, while an underfit model has low variance and high bias due to its simplistic nature. Both overfit and underfit models are undesirable and it is ideal to find a suitable trade-off between bias and variance (hence complexity) to yield a well fit model capable of adapting to different datasets [18].

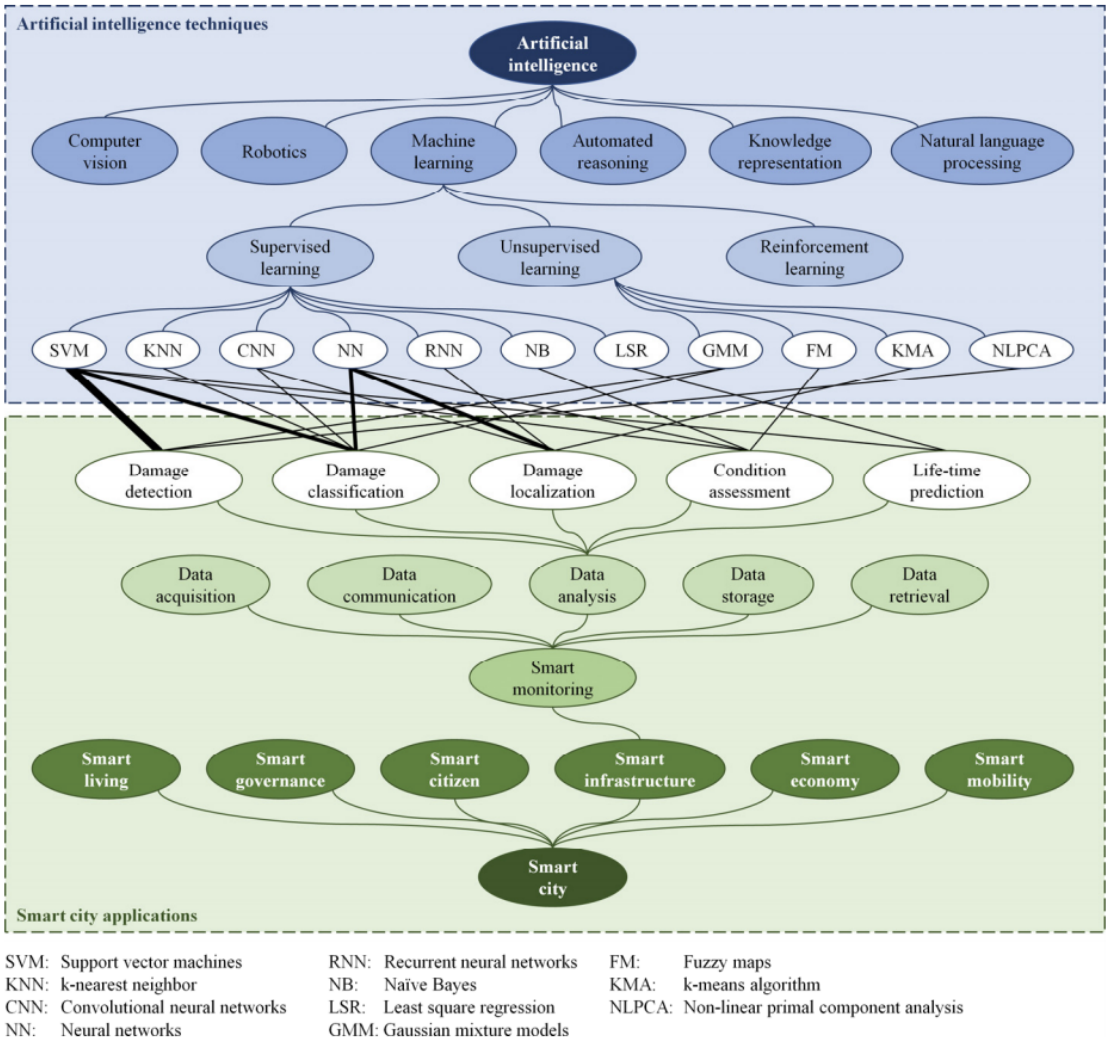


Figure 2.2: Available ML algorithms for smart monitoring [3]

The dataset/s to be used in the above experiments will be obtained through various open-source data collections available online. Therefore, data collection is not a part of this project. To ensure validity during the training of models, the data distribution will be split into three smaller datasets for training, validation and testing. The training set is used to train the models to fit the data and are evaluated against the validations set. The validation set being unseen, allows us to determine which models are generalising well to new examples. After the best model has been selected it is again tested on the test dataset as a final check on its generalisation ability. The training set accounts for 60% of the complete data set, while the validation and test sets account for 20% respecitvely.

2.3 Reward Hacking

The two unreliability factors discussed in the above experiments are concerned mainly with supervised learning models. A major reliability issue within reinforcement learning models is reward hacking. We will perform a systematic literature review on applications and known causes of unreliability due to reward hacking. Thereafter, we will explore potential solutions.

Label Bias and Environmental Datashift

Label Bias

Bias is the result of inadequate data where a certain group or class is favoured over another/others hence creating an overrepresentation [10] [16]. ML models trained using such datasets will acquire these underlying biases hence making incorrect predictions.

The following mathematical framework, developed by researchers at Google, can be used as a representation to undestand bias in data [10].

> **Assumption 1:** Presume fairness constraints are $c_1, ..., c_K$,
> where y_{true} is biased, namely $\mathbb{E}_{x\sim p}[\langle y_{true}(x), c_k(x)\rangle] = 0$ for $k \in [K]$
> Assuming that $\in_1, ..., \in_k \in \mathbb{R}$ exists and the observed biased label
> function, y_{bias} is the solution of the following equation.

$$\begin{aligned} &\arg\min_{\hat{y}:\chi\to[0,1]} \mathbb{E}_{x\sim P}[D_{KL}(\hat{y}(x)||y_{true}(x))] \\ &\text{s.t}\, \mathbb{E}_{x\sim P}[\langle \hat{y}(x), c_k(x)\rangle] = \in_k \\ &\text{for } k = 1, ..., K. \end{aligned} \tag{3.1}$$

D_{KL} signifies KL-divergence

In figure (3.1), the assumption is that y_{bias} is the label which is closest to y_{true} and achieves a measure of bias. In cases where data has been manually manipulated by human input, either consciously or subconsciously, this is deemed to be a reasonable assumption. The contiguity to y_{true} is given by the KL-divergence, which is used to

establish the notion of accurate labelling. The proposition in figure (3.2) is derived from the KL-divergence. (For complete proof of proposition, see [10])

Proposition 1: Presume Assumption 1 holds,
y_{bias} satisfies the following equation (for $x \in \chi$ and $y \in y$)

$$y_{bias}(y|x) \ \alpha \ y_{true}(y|x) \cdot \exp\left\{ -\sum_{k=1}^{K} \lambda_k \cdot c_k(x,y) \right\} \tag{3.2}$$
$$\textit{for some } \lambda_1, ..., \lambda_K \in \mathbb{R}$$

Now that y_{bias} is represented in terms of y_{true}, we can infer y_{true} in terms of y_{bias} as represented in Figure (3.3).

Corollary 1: Presume Assumption 1 still holds.
The unbiased label function, y_{bias} is nor represented as:

$$y_{true}(y|x) \ \alpha \ y_{bias}(y|x) \cdot \exp\left\{ \sum_{k=1}^{K} \lambda_k \cdot c_k(x,y) \right\} \tag{3.3}$$
$$\textit{for some } \lambda_1, ..., \lambda_K \in \mathbb{R}$$

Datashift

There may be situations where performance issues may not be apparent during training stages. They instead appear post-deployment, where training and deployment datasets can have irregularities. This is known as Environmental Datashift [16]. This calls into question whether the ML model is robust enough to generalise well to new samples beyond training or whether it tends to over-generalise to the training dataset, thus resulting in unreliability in the real world.

In [19], datashift has been mathematically modelled in two scenarios as follows:

Assuming classifier, $\mathcal{F}$ has been fairly trained with true labels, Z, at $t = 0$ (where $t = 0$ signifies time of training and $t = 1$). signifies a new timestep). Parameters

such as binary classification and probability of privelaged and distadvantaged class, ζ_A, ζ_B, ρ_A and ρ_B respectively, may change from $t = 0$ to $t = 1$.

The first scenario being when there is no access to the new labels at $t = 1$. Therefore the structure of the bias model at $t = 1$ is not apparent, but Z has changed. Consequently, ζ_A and ζ_B have also changed. We can assume:

$$\text{label shift: } P(B)_{t=0} = P(B)_{t=1}$$
$$\text{covariate shift: } P(B)_{t=0} \neq P(B)_{t=1}$$

The second scenario is when there is access to the new biased labels, in cases where deployment environment data is collected to validate the bias model at $t = 1$. There are two viable causes for shift, true distribution changes and bias model changes.

True distribution changes are similar to the first scenario where distribution, $\mathcal{D}_{t=1} \neq \mathcal{D}_{t=0}$. In bias model changes, the true distribution labels may be the same but label shift has occurred as a result of changing $\mathcal{G}$. This also means probability, p_A and/or p_B could have also changed, resulting in concept shift.

3.1 Dataset & Preprocessing

The predictive maintenance dataset [20] will be used to model bias and environmental datashift while classifying failures of an ioT gadget/s. During one week, maintenance data was collected from six devices every hour for 168 hrs. Therefore, this data set contains 1008 rows of data. Each cycle of data reading contains the following measurements:

Table 3.1: Measurements Dataset

Measurement	**Description**
Measurement Time	Time
Gadget ID	Device number
Vibration x sensor	Horizontal vibration
Vibration y sensor	Vertical vibration
Pressure sensor	Hose pressure
Temperature sensor	Internal temperature

The failures dataset contains the precise times each gadget failed. During the course of the week, 105 failures were recorded. The two datasets were combined

and additional labels were added (3.2) for use in training and prediction. The model was trained using *'Vibration y'*, *'Temperature 6hr Std'*, and *'Pressure 6hr Mean'* as feature labels, and predictions were tested using class label, *'Fail in 1hr'*, where positive classification of device failure occurs when the time remaining to device failure is less than one hour.

Table 3.2: Maniupulated data labels

Labels	**Description**
Temperature 6hr Std	Standard Deviation of last 6 measurements
Pressure 6hr Mean	Average of last 6 measurements
Fail in 1hr	If failure will occur within the next hour

The complete dataset was then split 70-30% for training and testing respectively. In addition, the training dataset was split into two further datasets, DF1 and DF2. DF1 contained all data from devices with 'Gadget ID' 1,2 and 3, while DF2 contained all data from devices with 'Gadget ID' 3,4 and 5. Consequently, the test dataset was also split into 'Sample 1' and 'Sample 2' in the same manner as DF1 and DF2.

Table 3.3: Training and Testing Datasets

Dataset	**Size**	**Description**
Full Train	685	70% of the complete dataset. Used for training model
Full Test	293	30% of the complete dataset. Used for testing model
DF1	339	Only samples from **Full Train** with Gadget ID 1,2 & 3
DF2	346	Only samples from **Full Train** with Gadget ID 4,5 & 6
Sample 1	150	Only samples from **Full Test** with Gadget ID 1,2 & 3
Sample 2	145	Only samples from **Full Test** with Gadget ID 4,5 & 6

3.2 Experimental Results

It has previously been assumed that bias occurs when one type of sample is over represented over another/others. By training a model with DF1 or DF2 dataset instead of the Full Train dataset, samples of certain devices have been completely overlooked and hence bias exists within DF1 and DF2. Using Table (3.4), we can compare the metrics of Full Train, DF1 and DF2 sets when tested against the

Full Test set. Although DF1 improves amongst most metrics of the Full Test, we observe a decrease in performance in the model trained with DF2. The increase in metrics of DF1 may suggest higher importance of those samples over DF2 but it is difficult to be certain due to the small dataset size (under 1000 samples).

Table 3.4: PDM Label Bias and Environmental Datashift Evaluation Results - SVM

Metric	**Datasets**						
	Full Train	DF1			DF2		
	Full Test	Full Test	Sample 1	DF2	Full Test	Sample 2	DF1
Accuracy	0.778	0.788	0.816	0.675	0.775	0.778	0.743
Precision	0.91	0.93	0.93	0.91	0.91	0.93	0.91
Recall	0.78	0.79	0.82	0.67	0.77	0.78	0.74
F1 Score	0.82	0.82	0.85	0.74	0.81	0.82	0.79

In the case of Environmental Datashift, *"modelers typically assume that training data is representative of the target population or environment where the model will be deployed" - [16]* We take a scenario where a model is trained on dataset DF1. Remembering that DF1 and Sample 1 datasets are only concerned with samples from Gadgets 1, 2 & 3, it is expected that this combination will offer the highest results. Experimental results in Table (3.4) and the plots in Figures (3.1) & (3.2) back this claim. Moving further, testing the model with Full Test yields the next best results as it contains a mix of already seen and completely new samples. Performance is affected severely when the model is tested using DF2 as the samples have irregularities when compared to training data. The same observations can be made when training the model on the DF2 dataset. This model is not robust enough to generalise to new samples, and solutions are offered in subsequent sections.

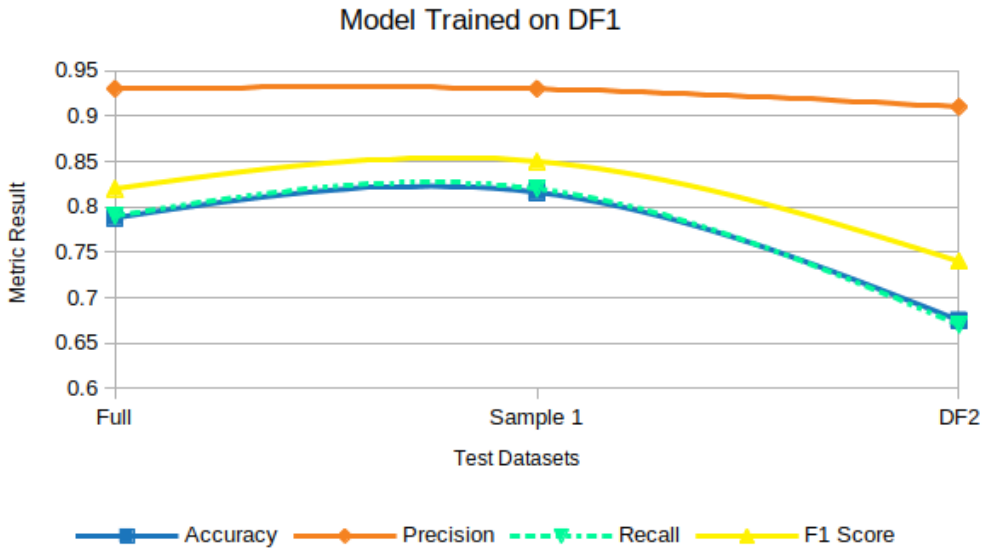


Figure 3.1: Test Datasets on DF1 metrics

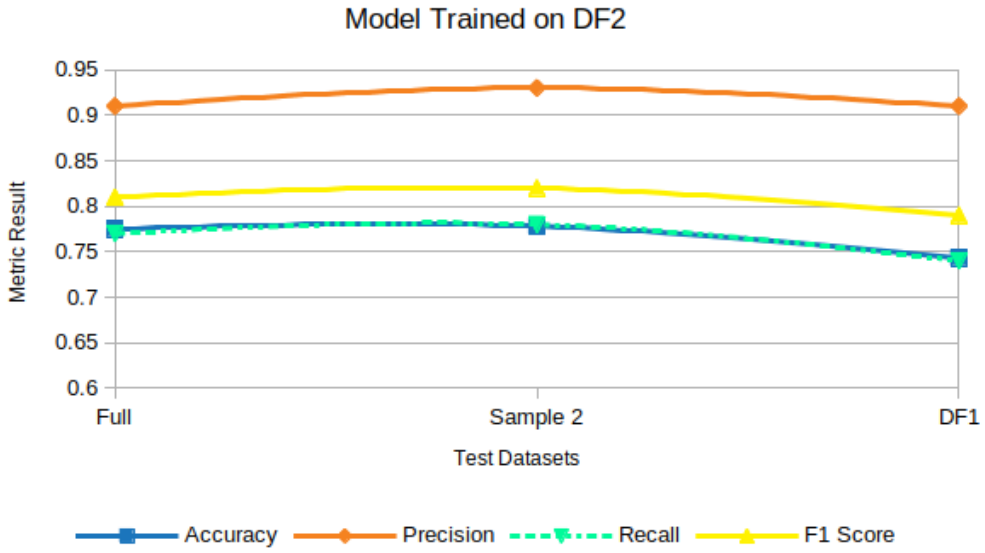


Figure 3.2: Test Datasets on DF2 metrics

3.3 Further Research

Research performed in [10], offers solutions to deal with biased datasets and compares these solutions across various datasets. The following datasets were used in this experiment:

- Bank Marketing - Prediction of subscription to bank product using age as the protected attribute
- Communities & Crime - Determine whether a community has a high crime rate. Four race features are used as protected attributes
- ProPublicas COMPAS - Predict recidivism based on a number of factors. Protected attributes are two race based and two gender biased
- German Statlog Credit Data - Predict good or bad credit rating. Two protected groups separated by age.
- Adult Income - Predict if individual income is greater than $50k. Two protected groups based on gender and another two groups based on race.

Table 3.5: Benchmark Fairness Tests on Multiple Datasets [10]

Table 1. **Experiment Results: Benchmark Fairness Tasks**: Each row corresponds to a dataset and fairness notion. We show the accuracy and fairness violation of training with no constraints (Unc.), with post-processing calibration (Cal.), the Lagrangian approach (Lagr.) and our method. **Bolded** is the method achieving lowest fairness violation for each row. All reported numbers are evaluated on the test set.

Dataset	Metric	Unc. Err.	Unc. Vio.	Cal. Err.	Cal. Vio.	Lagr. Err.	Lagr. Vio.	Our Err.	Our Vio.
Bank	Dem. Par.	9.41%	.0349	9.70%	.0068	10.46%	.0126	9.63%	**.0056**
	Eq. Opp.	9.41%	.1452	9.55%	.0506	9.86%	.1237	9.48%	**.0431**
	Eq. Odds	9.41%	.1452	N/A	N/A	9.61%	.0879	9.50%	**.0376**
	Disp. Imp.	9.41%	.0304	N/A	N/A	10.44%	.0135	9.89%	**.0063**
COMPAS	Dem. Par.	31.49%	.2045	32.53%	.0201	40.16%	.0495	35.44%	**.0155**
	Eq. Opp.	31.49%	.2373	31.63%	**.0256**	36.92%	.1141	33.63%	.0774
	Eq. Odds	31.49%	.2373	N/A	N/A	42.69%	**.0566**	35.06%	.0663
	Disp. Imp.	31.21%	.1362	N/A	N/A	40.35%	.0499	42.64%	**.0256**
Communities	Dem. Par.	11.62%	.4211	32.06%	.0653	28.46%	.0519	30.06%	**.0107**
	Eq. Opp.	11.62%	.5513	17.64%	**.0584**	28.45%	.0897	26.85%	.0833
	Eq. Odds	11.62%	.5513	N/A	N/A	28.46%	.0962	26.65%	**.0769**
	Disp. Imp.	14.83%	.3960	N/A	N/A	28.26%	.0557	30.26%	**.0073**
German Stat.	Dem. Par.	24.85%	.0766	24.85%	.0346	25.45%	.0410	25.15%	**.0137**
	Eq. Opp.	24.85%	.1120	24.54%	.0922	27.27%	.0757	25.45%	**.0662**
	Eq. Odds	24.85%	.1120	N/A	N/A	34.24%	.1318	25.45%	**.1099**
	Disp. Imp.	24.85%	.0608	N/A	N/A	27.57%	.0468	25.15%	**.0156**
Adult	Dem. Par.	14.15%	.1173	16.60%	.0129	20.47%	.0198	16.51%	**.0037**
	Eq. Opp.	14.15%	.1195	14.43%	.0170	19.67%	.0374	14.46%	**.0092**
	Eq. Odds	14.15%	.1195	N/A	N/A	19.04%	**.0160**	14.58%	.0221
	Disp. Imp.	14.19%	.1108	N/A	N/A	20.48%	**.0199**	17.37%	.0334

As shown in Table (3.4, all models were trained initially without any attempt to solve the bias problem (Unc.). Following on, Post-processing, Lagrangian Approach and a newly proposed method was applied and tested for fairness violation. It is observed that implementation of all approaches increases fairness as the violation decreases when compared to Uncalibrated results.

A study carried out by Samsung Research [11] performs experimentation on data bias and environmental datashift with Deep Neural Networks in a similar fashion to our own experiment in the previous section.

Bias was deliberately produced within three different datasets in this study investigate the algorithm's ability to unlearn bias. We are only concerned with the Dogs and Cats dataset, and IMDB Face datasets. The Dogs and Cats dataset consisted of 25,000 images which were then divided between TB1 and TB2 according to the colour of dogs/cats to produce bias as desired. This method of segregation ensures that any sample in TB2 will not be found in TB1 and vice-versa. The Test dataset consisted of 12,500 images.

The IMDB Face dataset was also segregated in a similar manner. The Test set accounted for 20% of the entire dataset, and the remaining was used for training. EB1 and EB2 datasets were formed based on age and gender of actors. There was no common data samples between any of these datasets.

Table 3.6: Evaluation Results on Dogs and Cats Dataset [11]

	Trained on TB1		Trained on TB2	
Method	TB2	Test	TB1	Test
Baseline	.7498	.9254	.6645	.8524
Gray†	.8366	.9483	.7192	.8687
BlindEye [1]	.8525	.9517	.7812	.9038
GRL [8]	.8356	.9462	.7813	.9012
BlindEye+GRL	.8937	.9582	.8610	.9291
Ours-adv	.8853	.9594	.8630	.9298
Ours	**.9029**	**.9638**	**.8726**	**.9376**

Table 3.7: Evaluation Results on IMDB Face Dataset [11]

Method	Trained on EB1		Trained on EB2	
	EB2	Test	EB1	Test
	Learn Gender, Unlearn Age			
Baseline	.5986	.8442	.5784	.6975
BlindEye [1]	.6374	.8556	.5733	.6990
Ours	**.6800**	**.8666**	**.6418**	**.7450**
	Learn Age, Unlearn Gender			
Baseline	.5430	.7717	.4891	.6197
BlindEye [1]	**.6680**	.7513	**.6416**	.6240
Ours	.6527	**.7743**	.6218	**.6304**

By observing results presented in Tables (3.6 & 3.7), the effects of environmental datashift become apparent. When trained on TB1/EB1, networks tested on the full Test sets always perform better as the network is unable to generalise well enough to TB2/EB2. However, the experiment also proves that methods can be implemented to claw back some lost performance. Examining the Dogs and Cats dataset, the baseline result shows approximately a 20% reduction in performance on networks tested on TB1/2 when compared to the Test set. All evaluated methods are able to improve the percentage difference between biased datasets and the Test set. The best method is able to bring the percentage difference down to approximately 5%.

3.4 Recommendations

3.4.1 Pre-processing

When it comes to creating ML models for fair and unbiased predictions, the first area of investigation should be the data. In [16] and accompanying material, the issue of inadequate data is presented where the limited availability of data on minor subsets creates a negative bias towards that sub-population. Issues of this nature arise when the model is tested on the broader dataset where the resulting performance is rated highly, and performance on individual subsets is not evaluated. Therefore, it is vital to measure performance on all sub-populations to determine and understand the true performance of the ML model.

Another method to diagnose and identify reliability concerns if the sub-populations of interest are initially unknown would be to cluster the input space into regions of high and low density. In low-density regions, we would then need to investigate the regions to identify any correlation with attributes of concern or importance.

In cases where the performance in any one of the sub-populations is not up to standard, the simple remedy to this issue would be collecting or using data that better represents sub-populations of interest. This could be the use of more data or the use of more quality data in general. Another solution is to try and modify the objective function to balance out the sub-populations. Therefore, the underperforming data is given a higher weight and/or decrease in the weight of data overrepresented..

When developing ML models that are invariant and can generalise well to shifts in data, reactive solutions are predominantly offered. This involves optimising the model for its intended environment by using unlabeled data samples from the target distribution. This optimisation is usually carried out by implementing various techniques such as reweighting in order to adapt to the new deployment environment. However, reactive methods are only valuable when the actual environment data is available for investigation.

In situations where we cannot anticipate this change in the environment, proactive solutions are preferred. Proactive solutions teach the model to protect from likely problematic shifts by forecasting possible shifts. In [4], a framework is presented to teach predictive models to 'transport' or adapt to dataset shift. This framework uses directed acyclic graphs (DAGs) to represent how data shifts from unstable to stable environments in the data generation process (DGP). Here X denotes the inputs, and T is prediction. The result, T given X is stable (doesn't change over

time). The selection variable, S indicates what is likely to change. The use of DAGs assists in the identification of shifts which need to be protected against.

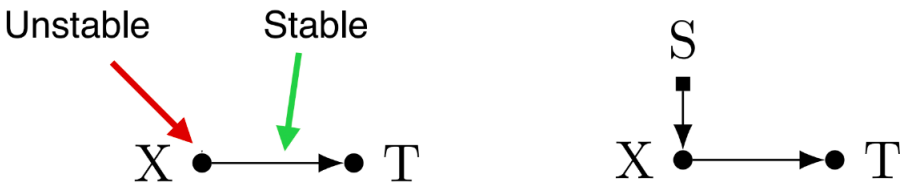


Figure 3.3: Representing datashift as graphs [4]

Once these invariant features have been identified, they can be fed through the 'surgery estimator' algorithm from [4]. The algorithm will then output the parts of the graph that should be learned and the parts which should be ideally ignored. The parts that should not be learned contain unstable relations. which will ultimately lead to the inability to generalise.

3.4.2 Post-processing

There may be scenarios where pre-processing approaches may be inapplicable or have been exhausted without successfully removing unwanted bias. Post-processing approaches are generally 'black-box' which signifies that it does not need access to the training process or any other internal, making them suitable in run-time settings.

While many post-processing methods are available, IBM Research has proposed a post-processing method that aims to increase group and individual fairness [21]. *Group fairness* is defined as the equal measure of segregation of the total population into groups such as gender or race. Individual fairness is the notion of individuals being treated fairly and is more demanding in terms of computation.

The method involves training an individual bias detector, $\hat{b}$, using dataset, $\{(x_1, \beta_1), ..., (x_m, \beta_m)\}$ once the classifier $\hat{y}$ has been trained. Alongside will be a validation dataset which contains no labels. Some of the samples in this dataset will have individual bias while others will not. The bias detector will generalise to new examples whose level of bias is unknown.

After the individual bias detector has been trained, we can apply the bias mitigation algorithm [21] in real-time. The algorithm is provided below:

Algorithm 1 Group and Individual Bias Mitigation Algorithm [21]

Given the classifier $\hat{y}$, which is trained on training set (x_i, d_i, y_i)
Given validation set $\{x_j | d_j = 0\}$
Compute individual bias, $\{b_{S,j} | d_j = 0\}$
if $b_{S,j} > \tau$ **then**
  $\beta_j \leftarrow 1$
else
  $\beta_j \leftarrow 0$
end
Formulate supplementary dataset, $\{(s_j, \beta_j) | d_j = 0\}$
Train individual bias detector, $\hat{b}$ on supplementary dataset
forall *run-time test samples* (x_k, d_k) **do**
  $\hat{y}_k \leftarrow \hat{y}(x_k, d_k)$
  if $d_k == 0$ **then**
    $\hat{b}_k \leftarrow \hat{b}(x_k)$
    if $\hat{b}_k == 1$ **then**
      $\hat{y}_k \leftarrow \hat{y}(x_k, 1)$
    else
      $\hat{y}_k \leftarrow \hat{y}_k$
    end
  else
    $\hat{y}_k \leftarrow \hat{y}_k$
  end
end

Suitable Algorithm Selection

Unreliable Machine Learning models can be the result of inadequate model assumptions where inappropriate or unsuitable algorithm/s have been used. The appropriacy of an algorithm is dependant on multiple factors. One such factor is the level of supervision required, which depends on the amount and type of data available. Another critical factor is the use case of the model and its intended outcomes. Generally, model parameters are curated for specific applications and will differ from other use cases. Therefore, it is important to make use of inductive bias [16] through suitable algorithm selection when developing reliable models.

In this chapter we will first train a number of models using different supervised learning algorithms before comparing respective evaluation metrics. Afterwards, research will be performed on algorithm selection for other use cases before final recommendations are presented.

4.1 Dataset & Preprocessing

The predictive maintenance dataset will be used again to classify failures of an IoT gadget. During one week, maintenance data was collected from six devices every hour for 168 hrs. Therefore, this data set contains 1008 rows of data.

This dataset has been split into two datasets for training and testing respectively. The training dataset will compromise of all the data collected from gadget IDs 1-4, leaving data from gadgets 5 and 6 for the test set. This will ensure that the trained models are tested on completely new data. For more information on the dataset and use case, please see [20].

4.2 Algorithms

The following section shortlists and describes algorithms that can be used to train a supervised model. Also discussed are the factors affecting the performance and reliability of these algorithms.

4.2.1 Support Vector Machines

Support Vector Machines (SVM) is a common supervised machine learning algorithm for both classification and regression tasks. A key attribute of SVMs is its high accuracy and precision in the segregation of classes. SVMs create n-dimensional hyperplanes to segregate datapoints into n number of classes/groups. The algorithm aims to achieve the maximum margin between support vectors (closest points), i.e. maximise the minimum margin.

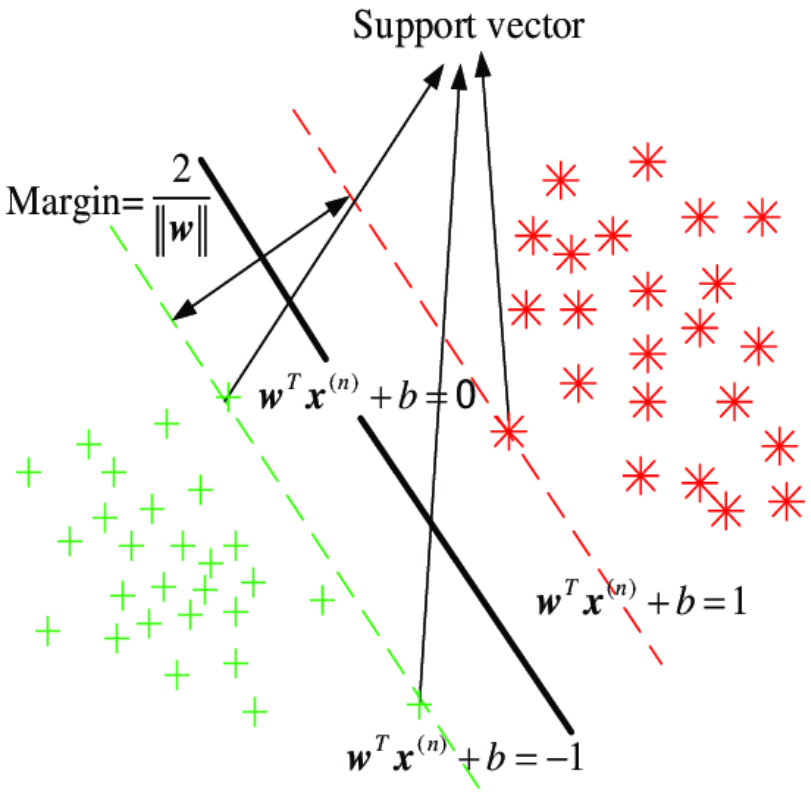


Figure 4.1: Support Vector Machine [5]

In the case where two classes can be linearly seperated, we consider the following as a representation of a dataset, S [22] [23].

$$S = \left\{ x_i \in \mathbb{R}^{1\times p}, y_i \in \{-1, 1\} \right\}_{i=1}^{n} \tag{4.1}$$

The values $\{-1, 1\}$ represent two classes of data, A and B,

$$y_i = \begin{cases} 1, & \text{if } i\text{-th sample} \in A \\ -1, & \text{if } i\text{-th sample} \in B. \end{cases} \tag{4.2}$$

The hyperplane can then be defined as F_0 in $\mathbb{R}^D$ space as,

$$F_0 = \left\{ x | f(x) = x\beta + \beta_0 = 0 \right\} \tag{4.3}$$

where, $\beta \in \mathbb{R}^D$ with norm $||\beta|| = 1$

For a new sample x^{new} which is not within dataset S, we can detemine a classification as,

$$y_{new} = \begin{cases} 1(\text{Class A}) , & \text{if } f(x^{new}) > 0 \\ -1(\text{Class B}) , & \text{if } f(x^{new}) < 0 \end{cases} \tag{4.4}$$

The performance of SVMs can be further improved by implementing kernel methods. Some popular kernel functions are:

$$\begin{aligned} \textbf{Polynomial: } & K(x_i, x_j) = (1 + \langle x_i, x_j \rangle)^d \\ \textbf{Radial Basis (RBF): } & K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|}{\sigma^2} \right) \\ \textbf{Neural Network: } & K(x_i, x_j) = \tanh(\langle x_i, x_j \rangle + b) \end{aligned}$$

A linear kernel will be used in this experiment.

4.2.2 k-Nearest Neighbours

k-Nearest Neighbours (k-NN) is a simple supervised machine learning algorithm used in both classification and regression problems. This approach classifies objects based on the computational distances or similarities between samples/values. The k-NN algorithm only requires tuning of a single parameter, k, which represents the amount of nearest samples within the neighbourhood [23]. The choice of k will affect

the algorithm's performance where a value too small would create higher variance, resulting in less stability. A larger k value will produce higher bias resulting in lower precision.

After the number of neighbours, k, has been selected, the distances between the query data point, x_q, and an arbitrary data point, x_i are to be determined. Most commonly used is the Euclidean distance (4.5), however, Manhattan distance (4.6) may also be applied [24].

$$d(x_q, x_i) = \sqrt{\sum_{i=1}^{m}(x_q - x_i)^2} \tag{4.5}$$

$$d(x_q, x_i) = \sum_{i=1}^{m}|x_r - x_i| \tag{4.6}$$

The resulting values are then sorted by distance from smallest to largest, and the first k entries are selected. In classification problems, the mode of k labels is returned, while the mean of k labels is returned in regression problems. The choice of parameter k can impact the model's reliability. Too small a value for k will result in higher variance, while a value too high will increase bias. This can be problematic when used on noisy datasets. Cross-validation can be used to determine the correct k value.

4.2.3 Random Forest

Random Forest is an ensemble machine learning method that creates multiple random decision trees and combines their respective votes (classification) or averages (regression) to improve prediction accuracy and fitting [25].

This algorithm is able to perform very well on huge datasets with thousands of variables. Moreover, it is able to achieve high accuracy scores without overfitting or the need for data pruning [26]. The rate of error experienced in random forests can come down to a number of factors. One such factor is the interrelationship between any two trees within the forest. If the interrelationship between two trees begins to increase, so will the error. Therefore, this algorithm is able to perform better in high dimensional trees as complexity increases.

For extensive information on the inner workings of Random Forests, see [27]. A brief algorithmic overview of random forests [25] is shown below:

Algorithm 2 Random Forest Algorithm for Classification and Regression [25]

1. for b = 1 to B:

   (a) Draw bootstrap sample $Z*$ of size N from the training data.
   (b) Develop random forest tree, T_B using bootstrapped data. Continuously repeating the subsequent process at each terminal node of the tree, until minimum node size, n_{min}

      i. Randomly select m variables from p variables
      ii. Choose the optimal variable/split-point among the m variables.
      iii. Split node into two child nodes

2. Output ensemble of trees $\{T_b\}_1^B$

To make a prediction at new point, x:

Classification: $\hat{C}_{rf}^B(x)$ = majority vote $\{\hat{C}_b(x)\}_1^B$, where $\hat{C}_b(x)$ is the class prediction of the bth random forest tree

Resgression: $\hat{f}_{rf}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$

4.2.4 Neural Networks

Neural Networks are a part of Deep Learning which in turn is a subset of Machine Learning. The network is made up of layers of neurons. The first layer, the input layer, receives input and the final layer, output layer, makes the prediction based on the computations taken place within the hidden layers. Neurons in adjacent layers are connected via channels and are assigned a numerical weight and threshold. If the output of any particular neuron exceeds the threshold value, that neuron is activated, and data is sent to the following layer within the network. Predictions are made based on the neuron with the highest value/probability at the output layer. Neural networks are self-adaptive through forward and backward propagation using real data

When it comes to reliability, it is essential to consider the amount of hidden layers and the amount of neurons within. Too few hidden layers would represent a linear problem, while additional layers create complexity hence becoming non-linear [28]. Thus, too few neurons would result in underfitting while too many neurons would not only cause overfitting but also significantly increase the training time.

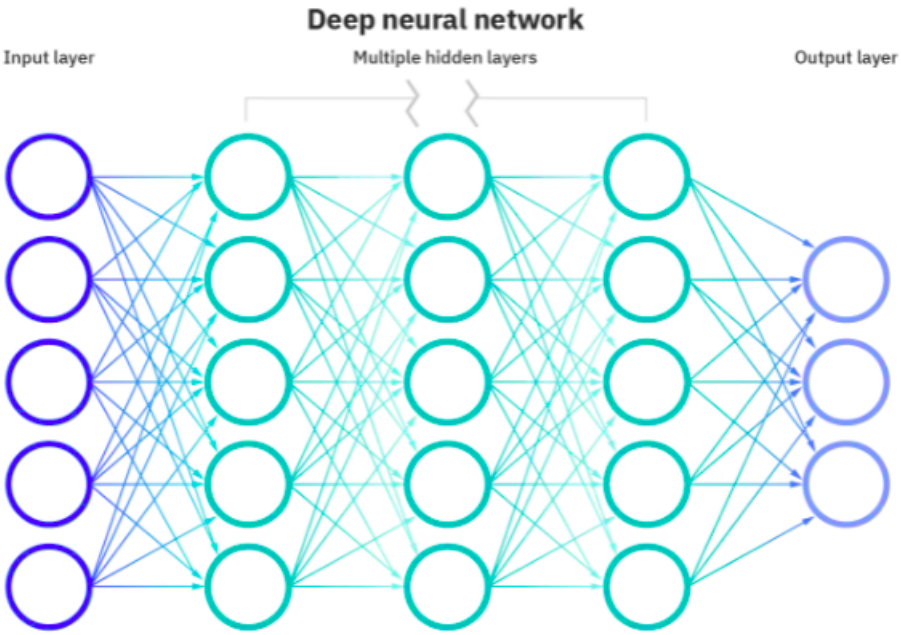


Figure 4.2: Neural Network [6]

Mathematically represented [6], the Neural Network follows equation (4.7) and activation is dependant on equation (4.8).

$$\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias \tag{4.7}$$

$$\text{threshold function} = \begin{cases} 1 & \text{if} \sum w_i x_i + bias \geq threshold \\ 0 & \text{if} \sum w_i x_i + bias < threshold \end{cases} \tag{4.8}$$

where:

- x_i is an input
- w_i is a weight
- Each neuron is assosicated with a numerical bias

4.3 Experimental Results

Building on top of the predictive maintenance project [20], we train models using the following algorithms and test for Precision, Recall, Accuracy, and AUC scores. The results are depicted in Figures (4.3, 4.4), and Tables (4.1, 4.2)

- Random Forests (RF)
- Logarithmic Regression
- Linear Regression
- k-Nearest Neighbours (knn)
- Neural Networks (nn)
- Support Vector Machines (SVM)
- Naive Bayes
- Stochastic Gradient Descent (SGD/clf)

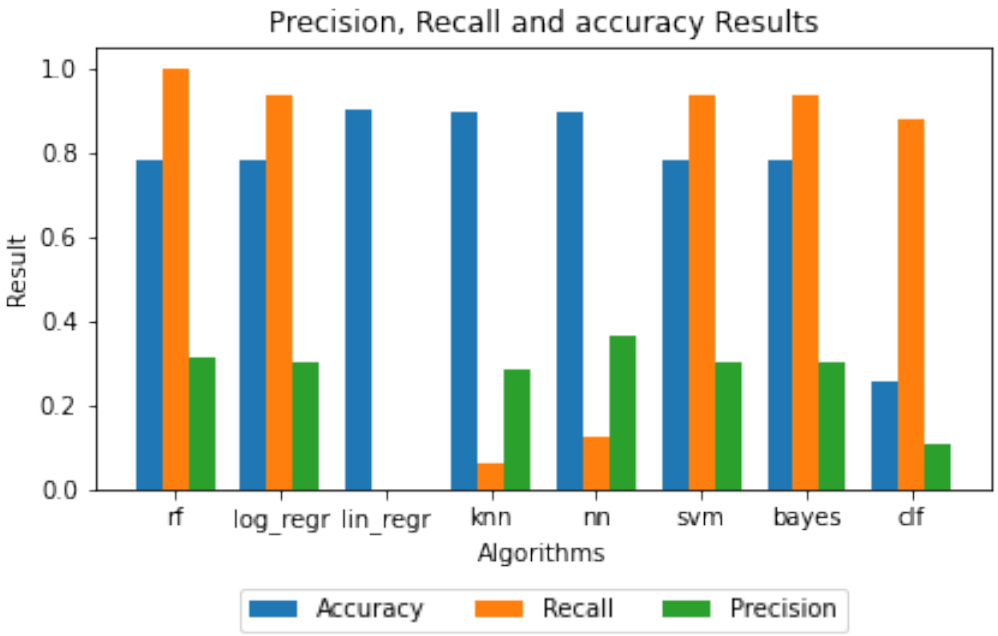


Figure 4.3: Precision, Recall and Accuracy Scores

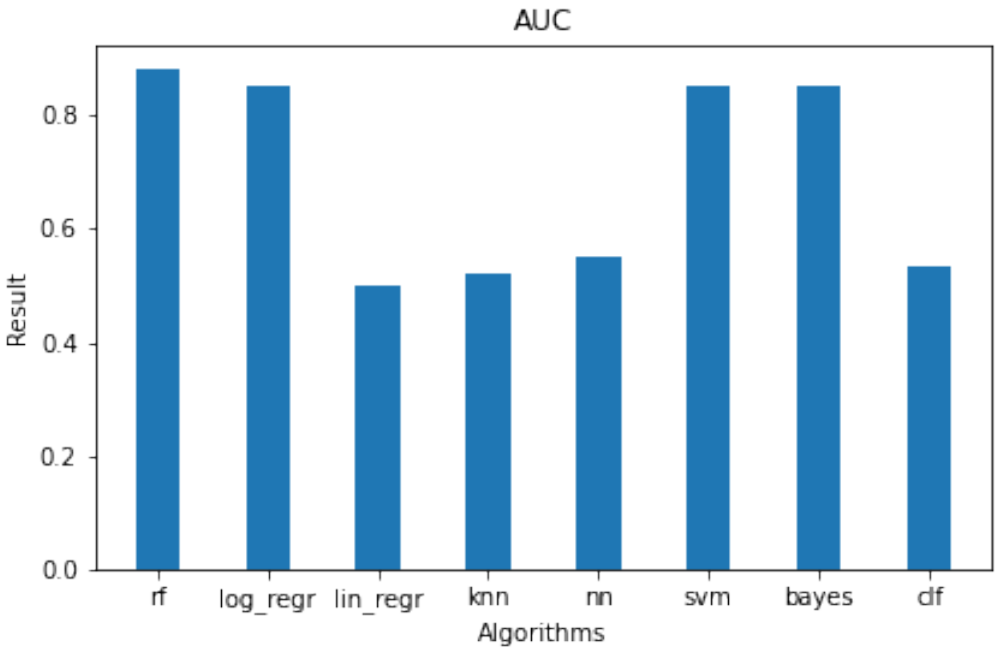


Figure 4.4: Area Under the Curve

Table 4.1: PDM Classification Metrics

Algorithm	Precision	Recall	Accuracy	AUC	F1
Random Forest	0.310680	1.0000	0.782875	0.879661	0.474074
Logarithmic Regression	0.300000	0.9375	0.779817	0.850106	0.454545
Linear Regression	0.000000	0.0000	0.902141	0.500000	0.000000
k Nearest Neighbours	0.285714	0.0625	0.892966	0.522775	0.102564
Neural Network	0.363636	0.1250	0.892966	0.550636	0.186047
SVM	0.303030	0.9375	0.782875	0.851801	0.458015
Naive Bayes	0.303030	0.9375	0.782875	0.851801	0.458015
CLF	0.104869	0.8750	0.256881	0.532415	0.187291

In Table 4.1, highlighted are the most optimal comparison scores between tested algorithms. The ensemble method, Random Forests, is evidently the most suited method to this predictive maintenance application. Linear Regression delivered the highest accuracy meaning it was able to make correct predictions the best. However, it did not perform well in the other metrics. This highlights that algorithm

evaluation cannot be determined based on accuracy alone and is supported by the study [29].

Table 4.2: PDM Confusion Matrix

Algorithm	**TP**	**FP**	**FN**	**TN**
Random Forest	224	71	0	32
Logarithmic Regression	225	70	2	30
Linear Regression	295	0	32	0
k Nearest Neighbours	290	5	30	2
Neural Network	288	7	28	4
SVM	226	69	2	30
Naive Bayes	226	69	2	30
CLF	56	239	4	28

To ensure reliability and robustness, the goals of the use case should be considered and may contradict performance metrics. In this scenario, a True Positive (TP) case where the predicted failures are actual failures should take precedence as this is the money saving factor. On the other hand, a False Negative (FN) case occurs when the model predicts a non-failure, but in reality, a failure has occurred. This would result in considerable financial penalties for the company due to downtime. Therefore, the highest amount of TP cases and the lowest amount of FN cases are desired.

4.4 Further Research

A separate study compares existing regression based machine learning algorithms on a similar but more complex predictive maintenance application, *'Prediction of Remaining Useful Lifetime (RUL) of Turbofan Engine using Machine Learning'* [7].

During this study, models were trained on a dataset obtained by NASA's data repository, where turbofan jet engines are run until failure. During operation, 21 sensor measurements are recorded against a time series per engine. Training and evaluation occur on four datasets, each containing data from 100-250 engines. This dataset is much more complex than the one used in the previous experiments, hence it is expected to see much more variance in the accuracy of the tested algorithms.

Listed below are machine learning algorithms used to predict RUL in this study:

- Linear Regression
- Decision tree
- Support Vector Machine
- Random Forest
- K-Nearest Neighbours
- The K Means Algorithm
- Gradient Boosting Method
- AdaBoost
- Deep Learning
- Anova

Like the Random Forest method, Gradien Boosting, AdaBoost, and Anova are also ensemble methods.

Table 4.3: RUL Algorithms Root Mean Square Error values [7]

Algorithm	***Data Set 1***	***Data Set 2***	***Data Set 3***	***Data Set 4***	***Mean***
Linear Regression	29.91	31.49	45.64	39.81	36.71
Decision Tree	28.48	34.52	27.74	45.91	34.17
SVM	48.17	31.12	61.53	34.65	43.86
Random Forest	24.95	29.64	30.55	33.79	29.73
KNN	30.79	34.79	34.44	44.70	36.18
K Means	78.30	90.19	72.92	95.45	84.21
Gradient Boost	27.45	33.35	31.78	39.30	32.97
Ada boost	28.82	33.84	30.91	39.16	33.18
Deep Learning	29.62	42.41	46.82	38.11	39.24
Anova	33.50	41.14	35.46	51.44	40.38

Figure 4.5: RUL Algorithms Root Mean Square Error Plots per data set [7]

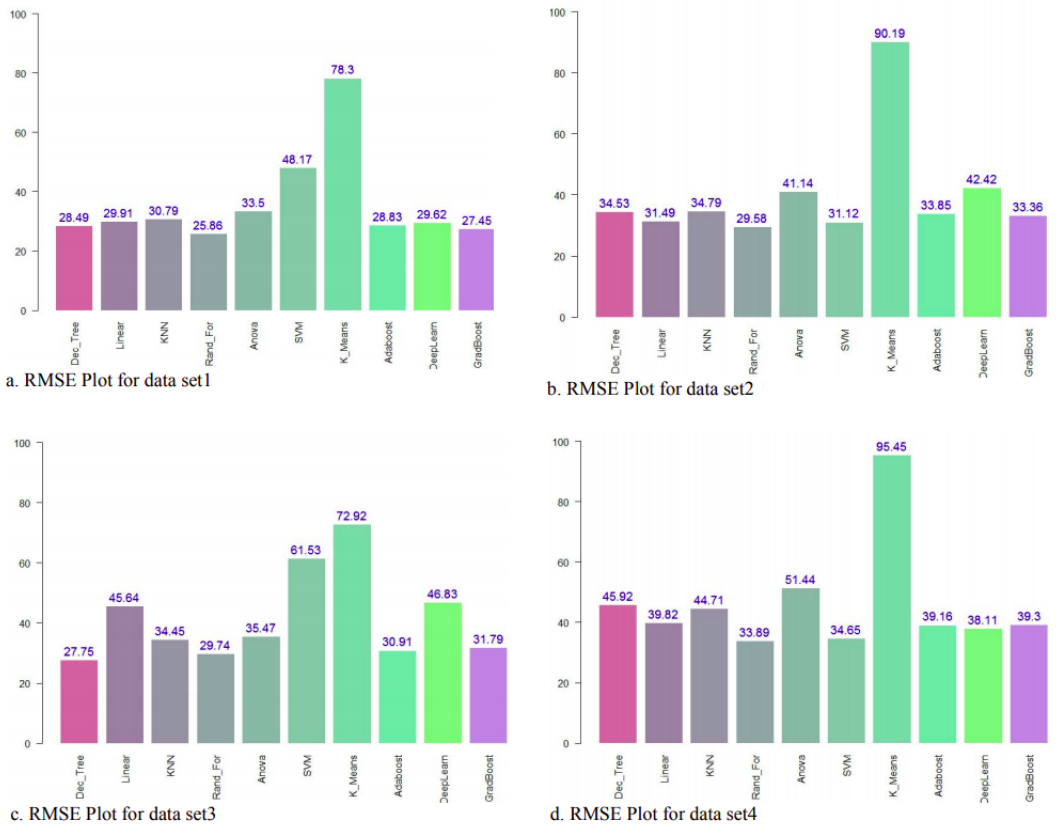


Table (4.3) and Figure (4.5) displays the RMSE values of all tested algorithms across all four datasets. As can be seen, the complexity of this problem results in fluctuation of evaluation scores, unlike our previous classification experiment. RMSE is a measure of the concentration of data around the line of best fit. Therefore, an algorithm with the lowest RMSE value is desired. In this application, the random Forest algorithm produced the lowest error value.

When comparing these results to those of our previous classification experiment, it is clear that ensemble methods perform the best, specifically, Random Forest. It should also be noted that SVM outperformed knn in the classification experiment but performed worse in the RUL regression study.

It is also important to consider other applications of AI/ML to better understand reliability of potential algorithms. Hence, moving away from predictive maintenance, [12] is a study of supervised learning techniques for the classification of different internal PD (partial discharge) patterns from a generator. Analysis was conducted

by considering 96 features for each PD pattern. A total of 9 algorithms under four learning were tested (shown below) where each was tested for accuracy, ROC, precision and recall.

- Functional Based Techniques
  - Logistic Regression
  - SVM
  - Multi-layer Perceptron (MLP) - Neural Network
- Probabilistic
  - Naive Bayes
- Decision Tree Models
  - J-48
  - Random Forest (RF)
  - Random tree
  - AdaBoost
- Nearest Neighbours
  - Instant Based-k (IBK)

Table 4.4: PD Algorithms accuracy, RUC, precision and recall scores [12]

Algorithm	**Accuracy (%)**	**ROC Area**	**Class 1**			**Class 2**			**Class 3**			**Class 4**		
			Precision	**Recall**	**Overall Rate (%)**	**Precision**	**Recall**	**Overall Rate (%)**	**Precision**	**Recall**	**Overall Rate (%)**	**Precision**	**Recall**	**Overall Rate (%)**
LR	92.4	0.973	0.885	0.920	90.2	1.000	0.929	96.3	0.909	1.000	95.2	0.900	0.818	85.7
SVM	**99.1**	**0.992**	1.000	1.000	**100.0**	1.000	1.000	**100.0**	0.968	1.000	98.4	1.000	0.955	97.7
MLP	**99.1**	0.990	1.000	1.000	**100.0**	1.000	1.000	**100.0**	0.968	1.000	98.4	1.000	0.955	97.7
Naïve Bayes	93.3	0.985	0.958	0.920	93.9	0.926	0.893	90.9	0.882	1.000	93.8	1.000	0.909	95.2
J-48	97.1	0.984	1.000	0.960	98.0	0.966	1.000	98.2	0.967	0.967	96.7	0.955	0.955	95.5
RF	99.0	0.991	1.000	1.000	**100.0**	1.000	1.000	**100.0**	0.968	1.000	98.4	1.000	0.955	97.7
Random Tree	95.2	0.967	1.000	1.000	**100.0**	0.962	0.893	92.6	0.882	1.000	93.8	1.000	0.909	95.2
Ada-Boost	97.1	0.992	1.000	0.960	98.0	0.933	1.000	96.6	0.967	0.967	96.7	1.000	0.955	97.7
IBK (k=3)	98.1	0.993	0.962	1.000	98.0	1.000	1.000	**100**	0.968	1.000	98.4	1.000	0.909	95.2

Generally, all algorithms have performed reasonably well. However, the Functional Based Techniques outperformed other techniques in accuracy scores. Naive Bayes performed the worst, having the lowest accuracy score and was unable to achieve a 100% rate in any of the four classes.

4.5 Recommendations

4.5.1 Model Assumptions

The results from previous sections highlight the importance of correct model specifications. Assumptions and/or expectations made in one use case most likely will not hold true in another application. Therefore, assumptions should be refined for every use case. A simple example provided in [16], involved the illogical use of a linear model on a complex (non-linear) problem.

To aid in the development of ML models, it is advised to research specific use case scenarios to find suitable algorithms through proven research into performance and reliability factors. In [30], comprehensive research and review has been performed on machine learning methods within predictive maintenance applications. The advantages of various algorithms have been identified in specific applications through experimentation. Therefore it would be beneficial to research for proven methodologies in any specific use case.

In [31], extensive comparisons are carried out over various studies on a range of classification applications and algorithms. Additionally, the study in, [32], provides advantages and disadvantages of some supervised learning techniques and suggests appropriate use cases for each algorithm.

4.5.2 Ensemble Methods

Ensemble methods are algorithms that combine multiple ML techniques to find the optimal predictive model in supervised and unsupervised ML problems. Generally, they perform much better than single algorithms yielding higher accuracy and reliability. The most popular techniques used are bagging for reduction in variance, boosting for reduction in bias, and stacking to improve the quality of predictions. Comparisons between various ensemble techniques have been studied in [33], [34] and [35]. As alluded to in Section (4.2.3), Random Forests is an example of a bagging method. AdaBoost, as the name suggests, is a popular boosting algorithm.

4.5.3 Hyperparamater Tuning

Another recommendation for improved reliability is hyperparameter tuning/optimisation. This can be used to aid in model design issues such as determining the appropriate number of trees in a random forest, depth of decision tree, amount of neural network layers and the amount of neurons in the layers. There exists many methods/algorithms for hyperparameter tuning/optimisation. Briefly described below are two algorithms, Bayesian Optimisation [36] and Hyperparameter Optimisation Machines (HOM) [37].

Algorithm 3 Brief Bayesian Optimisation Algorithm [36]

Input: Initial observation $D \equiv \{x_{1:p}, y_{1:p}\}$
Output: $\{x_n^*, y_n^*\}_{n=1}^T$
for $n = 1, 2, ..., T$ **do**
  Find $x_n^* = argmax_x \, \mathbb{E}(I(x)|GP(D))$ of (11)
  Calculate objective function: $y_n^* = f(x_n^*)$
  Enhance observation set $D = D \cup (x_n^*, y_n^*)$
end

Algorithm 4 Hyperparamater Optimisation Machines (HOM) [37]

Input: Hyperparamater space Λ observation history $\mathcal{H}$, transfer function $\mathcal{L}$, acquisition function a, surrogate model Ψ, tradeoff parameter α, hyperparam response function f (minimisation operation) total HOM iterations T
Output: Most optimal hyperparameter configuration
$\Lambda \leftarrow \emptyset, f - best \leftarrow \infty$

forall $t = 1, ..., T$ **do**
  Update: Fit Ψ to $\mathcal{H}$
  Predict: $\lambda \leftarrow \arg \, min_{\lambda\prime \in \Lambda}(1 - \alpha_t) \; \mathcal{L}(\lambda\prime, \Lambda t - 1) - \alpha_t a(\lambda\prime \Psi)$
  $\Lambda_t \leftarrow \lambda_{t-1} \cup \{\lambda\}$
  Calculate $f(\lambda)$
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{(\lambda, f(\Lambda))\}$

  if $f(\lambda) < f_{best}$ **then**
    $\lambda_{best}, f_{best} \leftarrow \lambda, f(\lambda)$
  end
end
return Λ_{best}

4.5.4 Evaluation Metrics

It is also essential to select appropriate evaluation metrics based on the model type and application. As observed in our earlier experiment in section (4.1), accuracy alone cannot be used to evaluate a model's reliability and performance. Accuracy is the fraction value of correct predictions made but does not indicate the relationships between other attributes of the model [29]. A *Confusion Matrix* is an N x N matrix specifying True Positives (TP), True Negatives (TN), False Positive (FP) and False Negative (FN) instances within the model. Other metrics can be derived using the confusion matrix [29]. Depending on the specific use case, a weight or cost can be assigned to each output of the confusion matrix to indicate importance. Table (4.5) below is a non-exhaustive list of some common evaluation metrics for supervised learning models. For evaluation metrics on unsupervised learning models, see [38].

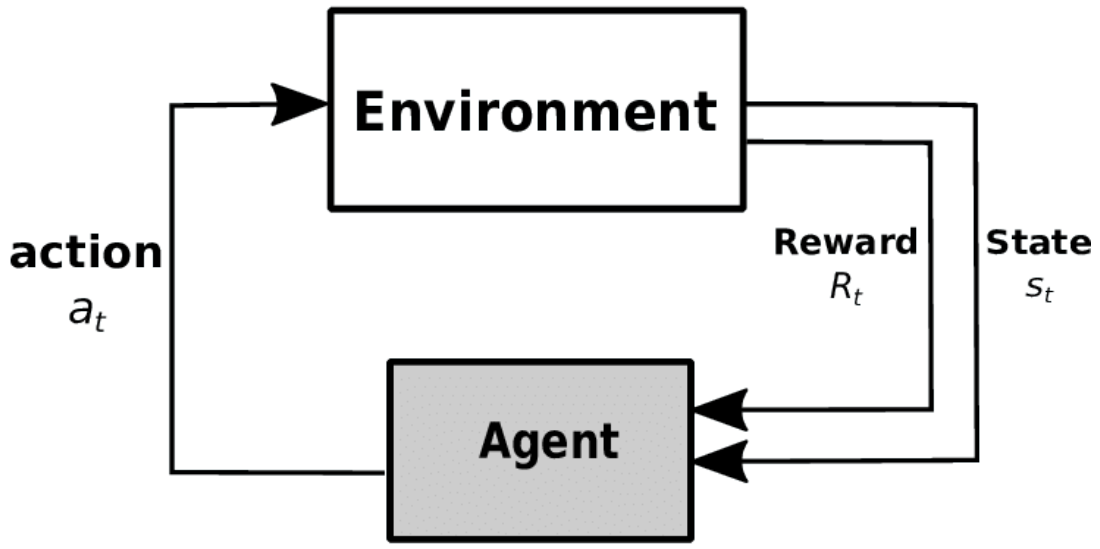
Table 4.5: Supervised Learning Evaluation Metrics

Metric	**Equation**	**Description**
Classification:		
Accuracy	$\frac{TN+TP}{TN+TP+FP+FN}$	Number of correct predictions made
Precision	$\frac{TP}{TP+FP}$	Rate of positive predictions that were infact positive
Recall	$\frac{TP}{FN+TP}$	Proportion of correctly classified positive cases
Specifity	$\frac{TN}{TN+FP}$	Number of correct negatice predictions
F1 Score	$\frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$	Harmonic mean of precision & recall
Regression:		
Root Mean Squared Error	$\sqrt{\frac{\sum_i^N = 1(Predicted_i - Actual_i)^2}{N}}$	How well regression line fits data points
R Squared	$1 - \frac{MSE(model)}{MSE(baseline)}$	Compares model to basic mean value
Adjusted R Squared	$1 - (1 - R^2)\left[\frac{n-1}{n-(k+1)}\right]$	Adjusts to changing features

Reward Hacking

Unlike Supervised and Unsupervised Learning methods, where predictions are made on underlying class types or patterns, respectively, Reinforcement Learning operates on a reward-action (trial and error) system where feedback is provided to determine the most optimal method to perform an action. As the agent takes actions, a_t to interact and manipulate the environment, state, S_t, and a reward function, R_t, are returned from the environment. The agent is then able to iterate through another cycle by performing another action based on the feedback and rewards received.

Figure 5.1: Reinforcement Learning and the Environment [8]



The Markov Decision Processes (MDP) can be used to represent an RL framework in terms of time steps. MDP is represented as a tuple, (S, a, P, R, γ) where:

- S is a set of states,
- a is a set of actions,
- P is the state trainstion probability matrix
- γ is a discount factor,
- R is a reward function

An action, a_t, taken under state, S_t leads to the following state, S_{t+1}.

$$s_t \xrightarrow{a_t} s_{t+1} \tag{5.1}$$

A reward, R_t, is then awarded for the current state based on the actions.

$$R_t(s_t, a_t) \tag{5.2}$$

However, RL agents are also susceptible to unreliabilties. The phenomenon of reward hacking occurs when the agent is able to maximise its reward in an unintended and undesirable way hence "is gaming the system". Not only is this undesired, but it can also be extremely dangerous in certain situations. Reward Hacking can be caused by Poor Feedback Loops, Goodhart's Law, Wireheading/Reward Function Tampering, Partially Observed Goals, and RL System Complexity. These issues are further discussed in Section 5.2.

5.1 Applications

RL approaches can also be applied to predictive maintenance applications to reduce critical downtime. As discussed earlier in Section (4.4), regressional models (as well as deep learning) are able to accurately make predictions on RUL for example. Nonetheless, such methods are unable to provide meaningful information or observations to aid the decision making process of maintenance operations due to complexities or unknowns in the environment [39].

An example of this RL scenario [39], where the objective function was to maximise the sensor lifetime. The reward function (5.3) was defined as such to manage the agent's actions:

$$R_t = \begin{cases} R_{Rpl}, & \text{if } S_t^i > 0, \beta > 0 \\ R_{Rpa}, & \text{if } S_r epresentedt^i > 0, \beta > 0 \\ R_{Exp}, & \text{if } S_t^i > 0 \\ R_{Frug}, & \text{if } S_t^i > 0, \beta > 0 \\ R_{Pen}, & otherwise \end{cases} \tag{5.3}$$

Another application of RL approaches is within traffic management systems (TMS). Traffic congestion increases environmental and noise pollution, fuel consumption, as well as operating costs. This study [40] aims to maximise the amount of vehicles through intersections while minimising the standard deviation of traffics jams. The following reward function was used to assess the agent's actions:

$$r_t = log_\delta(f(t)) \tag{5.4}$$

$$f(t) = \alpha \cdot (d_{ql}) + (1 - \alpha) \cdot (\tau^{tp}) \tag{5.5}$$

In order to exploit these reward functions, agents may start to behave irrationally. For example, the TMS may choose to give precedence to a particular queue within an intersection, or it may completely ignore the standard deviation if the throughput in a particular direction provides a better reward.

Similarly, if a higher reward is given to the predictive maintenance agent if it applies a 'repair' or 'replace' action, it may continuously flag a fault even though no faults exist.

In applications outside of predictive maintenance, a vacuum cleaning RL agent may choose to continuously eject dust to create a mess, and then clean it up to gain a reward [41]. Alternatively, an agent controlling a video game may choose to maximise the collection of points rather than completing the goal/mission [41]. A popular example of this was demonstrated in Super Mario World, where a glitch was taken advantage of to maximise points gained rather than playing the game as intended [42].

In the following two sections, we identify and discuss the causes of reward hacking and potential solutions as presented in studies carried out by researchers primarily from Google [17] [43].

5.2 Causes of Unreliability

The following sections describe some common causes of Reward hacking. There is no single cause or action that can cause an agent to game its reward system. Often a system may encounter a combination of these threats where the real problem lies deep within.

5.2.1 Poor Feedback Loops

If the objective function includes a parameter such as discount, γ, which can boost or diminish itself, another parameter becomes redundant. As a result, the objective function no longer behaves as it was intended [17]. The example provided in the previous section is an excellent example of this. The standard deviation was intended to equalise the importance of all queues. Nevertheless, it may be diminished by the strong feedback loop of vehicles passing through the intersections.

5.2.2 Goodhart's Law

> *"When a measure becomes a target, it ceases to be a good measure."*
> — *Charles Goodhart*

If the interrelationship between an objective function and its means or factors to successfully accomplishing a task is too high, that relationship will not hold under heavy optimisation [17]. Continuing with our TMS example, if the operation's success were only based on the throughput of vehicles across an intersection, the agent might choose to show a certain queue the green light forever. This would mean there is no queue of vehicles at this intersection, and the agent would keep collecting the maximum reward to keep the queue count at zero.

5.2.3 Wireheading & Reward Function Tampering

Wireheading is the ability of an intelligent agent to modify its reward sensors to ensure it always receives the maximum reward [44] [17]. In short, Wireheading occurs if the agent is no longer optimising the objective function but instead assumes control of the measuring system (reward process). Reward Function Tampering is defined as the agent's ability to modify or rewrite its reward function to yield maximum reward [43]. This encouragement to game the reward function results

in a deviation from the intended goal/s. At the present time, most RL agents are not yet intelligent enough to cause serious havoc in most applications however, some examples of Wireheading/Reward Tampering are presented in [43]. Concern arises in cases where humans have influence in the reward process (implicitly or explicitly) and can be a danger to themselves and others.

5.2.4 Partially Observed Goals

Unlike Wireheading and Reward Function Tampering, where the agent itself manipulates the reward function, Partially Observed Goals results from the designers inadequate reward function design. In turn, the inadequate design is the result of partial observations of the deployment environment and/or features that are difficult or impossible to measure. Therefore assumptions have to be made in an attempt to develop a reward function [17].

5.2.5 System Complexity

As the complexity of a program or system increases, so does the likelihood of bugs and glitches existing within the code [17]. Consequently, the chance of the reward system being taken advantage of increases. Despite the fact that such issues can be easily rectified, a simple bug or sensor fault in traffic cameras/sensors for a TMS system can and often will be taken advantage of by the agent.

5.3 Approaches for Prevention

5.3.1 Careful Engineering

In most cases, the most straightforward yet most tedious solution to reward hacking is careful engineering [17]. Performing formal verification or thorough practical testing is an easy way to iron out issues that may cause problems down the line. Cyberscuirty approaches such as sandboxing could also be beneficial as a means to segregate the agent from rewards.

5.3.2 Reward Strategies

5.3.2.1 Reward Capping

To prevent the agent performing high payoff and low probability approaches, reward capping can be implemented. This would disclose to the agent that performing these undesired actions will no longer provide a reward that is significantly larger than optimising the function in the desired manner [17].

5.3.2.2 Multiple Rewards

Multiple rewards functions can be used and combined by averaging or further optimisation to deter reward hacking. Each reward function could differ slightly, mathematically or by the objective function, ensuring that an invulnerability in one reward function does not impact the whole system [17]. Although it is unlikely, there is a chance that all reward functions could still be hacked.

5.3.2.3 Inverse Rewards

When an agent performs an action that takes a step towards the correct direction in terms of objective function, it is given a reward. The inverse could be applied as well. If the agent takes actions which move in the opposite direction to the objective function, it can be given a negative reward.

Furthermore, environmental datashift (as discussed in earlier sections) is also a factor that can cause unreliabilties in the reward function. If the reward function is designed only considering the training environment and its respective features, the agent's behaviour in different environments may be distasteful.

Inverse Reward Design involves determining the true reward function when given a proxy reward function, decision problem (MDP), and a set of possible reward functions. It can be represented by tuple $(R, \tilde{M}, \tilde{R}, \pi(\cdot|\tilde{r}, \tilde{M}), \tilde{r})$, with the goal being to recover r. For a detailed mathematical explanation of Inverse Reward Design see, [41].

Where:

- R is a space of possible reward functions,
- $\tilde{M}$ is the world model,
- $(-, \tilde{M}, \tilde{R}, \pi(\cdot|\tilde{r}, \tilde{M}))$ represents a partial RDP (reward Design Problem), P

5.3.2.4 History Based Rewards

The following claim has been made in [43]:

> *"A history-based reward function exists that avoids the RF-input tampering problem, if a deterministic (history-based) policy exists that reliably performs the task."*

This method is a viable option to prevent reward function tampering within RL systems that are competent in following the policy and accomplishing its required task. Here the reward function has complete access to historical actions and observations and is pushed to complete the intended task. However, this approach is susceptible to any intelligent agent that would have the ability to outsmart the reward function and has no preventative measures in place either.

5.3.2.5 Belief Based Rewards

Another viable method to solve reward function tampering is to base the rewards on the agent's hidden beliefs instead of history, as discussed previously. The predicitve model, $P(O_{t+1:m} \mid O_{1:t}, A_{1:t}, \pi)$, may be used by a predictive model. Future predictions, $O_{t+1:m}$, under the policy π and when given the observations and actions from the past. The relevant parts of the past are summarised by the belief state, B_t. Therefore, $P(O_{t+1:m} \mid O_{1:t}, A_{1:t}, \pi) = P(O_{t+1:m} \mid B_t, \pi)$. B_t can then be fed straight into a belief based reward function, $R_t = R(B_t; \theta^R)$.

5.3.3 Model Lookahead

A viable solution to prevent the model/agent from replacing its reward function (Reward Tampering) is to implement Model Lookahead. A model based RL system uses a model to develop a strategy for future actions by assessing the states a series of actions would lead to. A reward can be provided according to the model's anticipated state/s rather than providing a reward with respect to the current state [17]. As a result, the agent will not be able to benefit from reward hacking as these exploitations will most likely not reach the anticipated states. To represent Mathematically:

$$\begin{aligned} \text{current reward function:} \quad & \theta_k^R \\ \text{simulated future trajectories} \quad & k_{k+1}, ...S_m \\ \text{at time } k, \quad & R_t^k = R(S_t; \theta_k^R) \end{aligned}$$

This notion continues in a similar study (see for complete mathematical breakdown) [9], where three agent definitions are proposed, in Figure (5.2) and Table (5.1). Hedonistic value functions are calculated using the future utility function, u_{t+1}, instead of the current utility function, u_t. Therefore, these value functions promote self-modification. The modification of utility functions for Ignorant and Realistic value functions only affect modifications of future versions of that model.

Figure 5.2: Self-modification value functions [9]

Definition 10 (Hedonistic value functions)**.** A *hedonistic agent* is a policy optimising the *hedonistic value functions*:

$$V^{\mathrm{he},\pi}(æ_{<t}) = Q^{\mathrm{he},\pi}(æ_{<t}\pi(æ_{<t})) \tag{3}$$

$$Q^{\mathrm{he},\pi}(æ_{<t}a_t) = \mathbb{E}_{e_t}[u_{t+1}(ǽ_{1:t}) + \gamma V^{\mathrm{he},\pi}(æ_{1:t}) \mid ǽ_{<t}ǎ_t]. \tag{4}$$

Definition 11 (Ignorant value functions)**.** An *ignorant agent* is a policy optimising the *ignorant value functions*:

$$V_t^{\mathrm{ig},\pi}(æ_{<k}) = Q_t^{\mathrm{ig},\pi}(æ_{<k}\pi(æ_{<k})) \tag{5}$$

$$Q_t^{\mathrm{ig},\pi}(æ_{<k}a_k) = \mathbb{E}_{e_t}[u_t(ǽ_{1:k}) + \gamma V_t^{\mathrm{ig},\pi}(æ_{1:k}) \mid ǽ_{<k}ǎ_k]. \tag{6}$$

Definition 12 (Realistic Value Functions)**.** A *realistic agent* is a policy optimising the *realistic value functions*:[4]

$$V_t^{\mathrm{re},\pi}(æ_{<k}) = Q_t^{\mathrm{re}}(æ_{<k}\pi(æ_{<k})) \tag{7}$$

$$Q_t^{\mathrm{re}}(æ_{<k}a_k) = \mathbb{E}_{e_k}\left[u_t(ǽ_{1:k}) + \gamma V_t^{\mathrm{re},\pi_{k+1}}(æ_{1:k}) \mid ǽ_{<k}ǎ_k\right]. \tag{8}$$

Table 5.1: Self-modification value functions [9]

	Utility	Policy	Self-mod.	Primary self-mod. risk
Q^{he}	Future	Either	Promotes	Survival agent
Q^{ig}	Current	Current	Indifferent	Self-damage
Q^{re}	Current	Future	Demotes	Resists modification

5.3.4 Adversarial Methods

RL systems are always looking for ways to game the reward function to obtain a high reward while disregarding the intended behaviour for the use case. Consequently, the relationship between the agent and reward function can be described as adversarial [17]. Generally, the RL system is a dynamic agent capable of adapting to its environment or changing its actions and even source code. On the other hand, the reward function is a static object with no means to protect itself from the agents exploitations.

To combat this, the reward function could have its own agent capable of exploring and adapting to its environment to prevent the primary agent from trying to game it. So the reward agent would search for states where the primary agent would claim a high reward, but the human/developer would label as a low reward due to undesired behaviour.

Another technique is Adversarial Blinding, where the agent becomes oblivious (blind) to particular variables. In such cases, the agent would not be able to completely understand its environment and eradicates its ability to understand how the reward is generated. If the agent cannot understand how the reward is generated, the likeliness of it being able to hack the reward function greatly diminishes.

5.3.5 Tripwires

Tripwires are a technique akin to a security system. The main goal of tripwires is to alert the human that the agent is attempting to exploit the reward function [17]. Ideally, the technique would also stop the agent before the reward function has been taken advantage of. Despite the fact that this is not a solution to reward hacking, it will reduce the risk of harmful actions but also act as a measurable quantity or diagnostic which can be used to improve the objective function further.

5.3.6 Algorithmic Approaches

Researchers in [45] have identified drawbacks in some of the aforementioned approaches for the prevention of reward hacking. For instance reward capping can weaken the influence of exploitations but also weaken the performance of the RL agent as a whole. On the other hand, traditional multi-step approaches are also a potential solution where the agent pays attention to for a longer amount of time (instead of a single time-step).

Subsequently, researchers proposed a new 'novel multi-step approach' that uses a new return function defined:

$$\hat{G}_t^{(n)} = \frac{1}{n} \sum_{k=1}^{k=n} G_{t+k-1} \tag{5.6}$$

This function is the average of n standard return functions $G_t, G_{t+1}, \ldots G_{t+n-1}$, and intends to change the discount of future rewards and lessens the effect of the immediate reward. For a detailed breakdown of this approach, it is recommended to visit [45]

For further reading, algorithmic approaches for wireheading and reward function tampering are discussed in [44] and [43], respectively.

Conclusion

In this capstone project, we have discussed the importance of reliability in Artificial Intelligence and especially so in high-risk applications. Specifically, the reliability factors of label bias and environmental datashift, suitable algorithm selection, and reward hacking were investigated and recommendations provided. To summarise, we first explored bias and how it can arise within datasets and deployment environment. Bias was then replicated through experimentation on a predictive maintenance application. Proposed solutions were split into two categories; pre-processing and post-processing. Pre-processing methods are primarily concerned with data collection and preventing the underrepresentation of groups. Reactive and proactive methods were also discussed. Post-processing approaches are also available, but it should be noted that it is almost impossible remove bias from data altogether.

Secondly, suitable algorithm selection factors were studied. The goal was to understand how algorithms can be made more robust for all applications. Therefore, reliability and performance variables for algorithms were identified and discussed. Comparison between our experimentation on a predictive maintenance dataset and various other studies proved that ensemble methods are the better options than standalone algorithms, which tend to vary by application. Furthermore, the reliability of algorithms and algorithm selection can also be improved with hyperparameter optimisation and proper consideration of evaluation metrics based on the use case.

Finally, a systematic literature review on reward hacking highlighted the importance of AI safety when powerful agents are involved. Reward hacking can be caused by issues such as RF tampering and system complexity. It is also common for a number of these identified issues to occur in combination hence increasing the need for controls. A number of methods, such as reward strategies and algorithmic approaches, have been offered to resolve reward hacking.

References

[1] Q. Chen, W. Wang, F. Wu, S. De, R. Wang, B. Zhang, and X. Huang, "A survey on an emerging area: Deep learning for smart city data," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 3, no. 5, pp. 4392–410, 2019.

[2] B. Mohapatra, "Machine learning applications to smart city," *ACCENTS Transactions on Image Processing and Computer Vision*, vol. 5, pp. 1–6, 02 2019.

[3] D. Luckey, H. Fritz, D. Legatiuk, K. Dragos, and K. Smarsly, "Artificial intelligence techniques for smart city applications," 08 2020.

[4] A. Subbaswamy, P. Schulam, and S. Saria, "Preventing failures due to dataset shift: Learning predictive models that transport," 2019.

[5] S. Chen, B. Shen, X. Wang, and S.-J. Yoo, "Geo-location information aided spectrum sensing in cellular cognitive radio networks," *Sensors*, vol. 20, p. 213, 12 2019.

[6] IBM, "Neural networks." https://www.ibm.com/cloud/learn/neural-networks, 2020.

[7] V. Mathew, T. Toby, V. Singh, B. M. Rao, and M. G. Kumar, "Prediction of remaining useful lifetime (rul) of turbofan engine using machine learning," in *2017 IEEE International Conference on Circuits and Systems (ICCS)*, pp. 306–311, 2017.

[8] R. Amiri, H. Mehrpouyan, L. Fridman, R. K. Mallik, A. Nallanathan, and D. Matolak, "A machine learning approach for power allocation in hetnets considering qos," *2018 IEEE International Conference on Communications (ICC)*, 2018.

[9] T. Everitt, D. Filan, M. Daswani, and M. Hutter, "Self-modification of policy and utility function in rational agents," *CoRR*, vol. abs/1605.03142, 2016.

[10] H. Jiang and O. Nachum, "Identifying and correcting label bias in machine learning," Jan 15 2019.

[11] B. Kim, H. Kim, K. Kim, S. Kim, and J. Kim, "Learning not to learn: Training deep neural networks with biased data," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9004–9012, 2019.

[12] H. Herath, J. Kumara, M. Fernando, K. Bandara, and I. Serina, "Comparison of supervised machine learning techniques for pd classification in generator insulation," in *2017 IEEE International Conference on Industrial and Information Systems (ICIIS)*, pp. 1–6, 2017.

[13] I. Saif and B. Ammanath, "trustworthy ai is a framework to help manage unique risk.." MIT Technology Reviw, 2020.

[14] R. Moutafis, "How bad facial recognition software gets black people arrested.." towardsdatascience, 2020.

[15] B. E, L. R. Flaih, D. Yuvaraj, S. K, A. Jayanthiladevi, and T. S. Kumar, "Use case of artificial intelligence in machine learning manufacturing 4.0," in *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, pp. 656–659, 2019.

[16] S. Saria and A. Subbaswamy, "Tutorial: Safe and reliable machine learning." ACM Conference on Fairness, Accountability, and Transparency (FAT* 2019)., 2019.

[17] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Man, "Concrete problems in ai safety," Jul 25 2016.

[18] D. Jedamski, "Bias/variance tradeoff - applied machine learning: Foundations. linkedin learning.." Available at `https://www.linkedin.com/learning/appliedmachine-learning-foundations/bias-variance-tradeoff?u=2129308`, 2019.

[19] D. J and M. S, "Label bias, label shift: Fair machine learning with unreliable labels." Consequential Decisions in Dynamic Environments, 2020.

[20] M. Ahonen and E. Lahtinen, "Predictive maintenance tutorial." https://github.com/Unikie/predictive-maintenance-tutorial, 2020.

[21] P. K. Lohia, K. N. Ramamurthy, M. Bhide, D. Saha, K. R. Varshney, and R. Puri, "Bias mitigation post-processing for individual and group fairness." https://www.ibm.com/downloads/cas/WM4MWDOE, 2018.

[22] G. A. Susto, A. Schirru, S. Pampuri, D. Pagano, S. McLoone, and A. Beghi, "A predictive maintenance system for integral type faults based on support vector machines: An application to ion implantation," in *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 195–200, 2013.

[23] G. Susto, A. Schirru, S. Pampuri, S. McLoone, and A. Beghi, "Machine learning for predictive maintenance: A multiple classifiers approach," *IEEE Transactions on Industrial Informatics*, vol. 11, pp. 812–820, June 2015.

[24] D. Bhalla, "K nearest neighbour." Listen Data, Data Science, 2017.

[25] Y. Yang, "Random forests." Mathematics and Statistics - McGill University.

[26] J. K. Jaiswal and R. Samikannu, "Application of random forest algorithm on feature subset selection and classification and regression," in *2017 World Congress on Computing and Communication Technologies (WCCCT)*, pp. 65–68, 2017.

[27] G. Biau, "Analysis of a random forests model," *J. Mach. Learn. Res.*, vol. 13, p. 10631095, Apr. 2012.

[28] A. Azadeh and Behshtipour, "The effect of neural network parameters on the performance of neural network forecasting," in *2008 6th IEEE International Conference on Industrial Informatics*, pp. 1498–1505, 2008.

[29] R. Choudhary and H. K. Gianey, "Comprehensive review on supervised machine learning algorithms," in *2017 International Conference on Machine Learning and Data Science (MLDS)*, pp. 37–43, 2017.

[30] T. P. Carvalho, F. A. A. M. N. Soares, R. Vita, R. da P. Francisco, J. P. Basto, and S. G. S. Alcal, "A systematic literature review of machine learning methods applied to predictive maintenance," *Computers and Industrial Engineering*, vol. 137, p. 106024, 2019.

[31] C. Zhang, C. Liu, X. Zhang, and G. Almpanidis, "An up-to-date comparison of state-of-the-art classification algorithms," *Expert Syst. Appl.*, vol. 82, p. 128150, Oct. 2017.

[32] A. Singh, N. Thakur, and A. Sharma, "A review of supervised machine learning algorithms," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 1310–1315, 2016.

[33] S. Ali, S. S. Tirumala, and A. Sarrafzadeh, "Ensemble learning methods for decision making: Status and future prospects," in *2015 International Conference on Machine Learning and Cybernetics (ICMLC)*, vol. 1, pp. 211–216, 2015.

[34] P. Kumari, P. K. Jain, and R. Pamula, "An efficient use of ensemble methods to predict students academic performance," in *2018 4th International Conference on Recent Advances in Information Technology (RAIT)*, pp. 1–6, 2018.

[35] Y.-S. Dong and K.-S. Han, "A comparison of several ensemble methods for text categorization," in *IEEE International Conference onServices Computing, 2004. (SCC 2004). Proceedings. 2004*, pp. 419–422, 2004.

[36] T. T. Joy, S. Rana, S. Gupta, and S. Venkatesh, "Hyperparameter tuning for big data using bayesian optimisation," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, pp. 2574–2579, 2016.

[37] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, "Hyperparameter optimization machines," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 41–50, 2016.

[38] J.-O. Palacio-Nio and F. Berzal, "Evaluation metrics for unsupervised learning algorithms," 2019.

[39] K. S. Hoong Ong, D. Niyato, and C. Yuen, "Predictive maintenance for edge-based sensor networks: A deep reinforcement learning approach," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pp. 1–6, 2020.

[40] H. Joo, S. H. Ahmed, and Y. Lim, "Traffic signal control for smart cities using reinforcement learning," *Computer Communications*, vol. 154, pp. 324–330, 2020.

[41] D. Hadfield-Menell, S. Milli, P. Abbeel, S. J. Russell, and A. D. Dragan, "Inverse reward design," *CoRR*, vol. abs/1711.02827, 2017.

[42] "Snes super mario world (usa) "arbitrary code execution"." Tool-assisted movies, http://tasvideos.org/2513M.html., 2014.

[43] T. Everitt and M. Hutter, "Reward tampering problems and solutions in reinforcement learning: A causal influence diagram perspective," *CoRR*, vol. abs/1908.04734, 2019.

[44] T. Everitt and M. Hutter, "Avoiding wireheading with value reinforcement learning," *CoRR*, vol. abs/1605.03143, 2016.

[45] Y. Yuan, L. Y. Zhu, Z. Gu, X. Deng, and Y. Li, "A novel multi-step reinforcement learning method for solving reward hacking," *Applied Intelligence*, vol. 49, pp. 2874–2888, 08 2019. Copyright - Applied Intelligence is a copyright of Springer, (2019). All Rights Reserved; Last updated - 2019-12-11.

Appendix

A.1 Communication Log

Project Title:	**Reliability of Artificial Intelligence**		
Student Name:	**Ajal Singh**	**Supervisor Name:**	**Diep Nguyen**
Date	**Event**	**Topic of Communication**	**Outcome**
18/1/21	Email	ERP Feedback	Feedback provided
21/1/21	Email	Experimentation scenarios discussed	Supervisor suggested cybersecurity
15/2/21	Email	Request change of focus applications	Supervisor accepted concerns and suggested new focus points
23/2/21	Email	Discussion about experimentation and plagiarism concerns	concerns settled
25/3/21	Email	Capstone Report marking questions	Questions answered
6/4/21	Email/Teams	Questions about evaluation metrics	Resources provided
13/4/21	Teams	First draft submitted for review	Pending Supervisor feedback
2/5/21	Teams	First draft feedback and further questions	Supervisor provided feedback and answered questions
5/5/21	Teams	Second draft feedback and further questions	Supervisor provided feedback and answered questions
27/5/21	Teams	Third draft feedback and further questions	Supervisor provided feedback and answered questions

A.2 Experiment Code

A.2.1 Label Bias and Environmental Datashift Experiment

https://github.com/ajalsingh/capstone/blob/master/predictive_maintenance_experiements/experiments/IoT/bias_datashift.ipynb

A.2.2 Suitable Algorithm Selection Experiment

https://github.com/ajalsingh/capstone/blob/master/predictive_maintenance_experiements/experiments/IoT/algorithm_selection.ipynb