University of Technology Sydney
Faculty of Engineering and Information Technology

Improving the Reliability of Artificial Intelligence

Capstone Report

Ajal Singh

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

May 17, 2021

Contents

List of Figures

List of Tables

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.

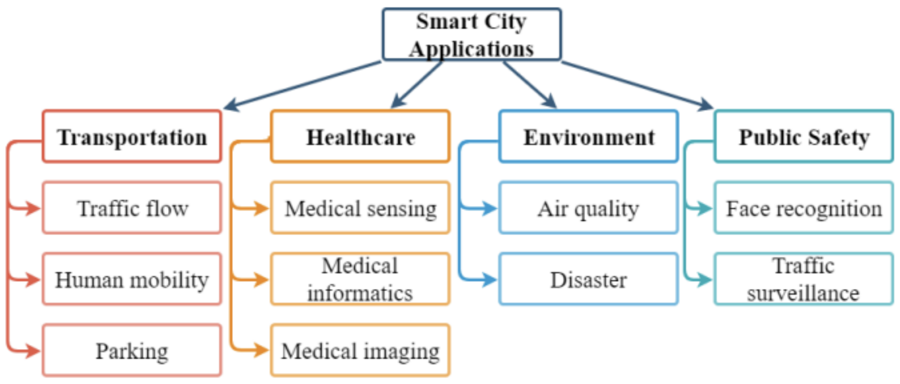


Figure 1.1: Smart City Artificial Intelligence Applications [1]

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 [8].

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 [9]. 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. Cancer diagnosis systems trialled in the US are failing to detect cancer in patients in differing hospitals and/or countries which may result 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* [10]. 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. Down-time 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 [10].

1.3 Project Contextualisation

A tutorial presented by Suchi Saria and Adarsh Subbaswamy of John Hopkins University [11] 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 the use of inadequate data, changes in training and deployment environments, and model misspecification. These aforementioned causes will form the basis of this research project.

Another reliability issue is discussed in a separate paper, *Concrete Problems in AI Safety* [12] 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.

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 Section 1. The main factors studied in this project are:

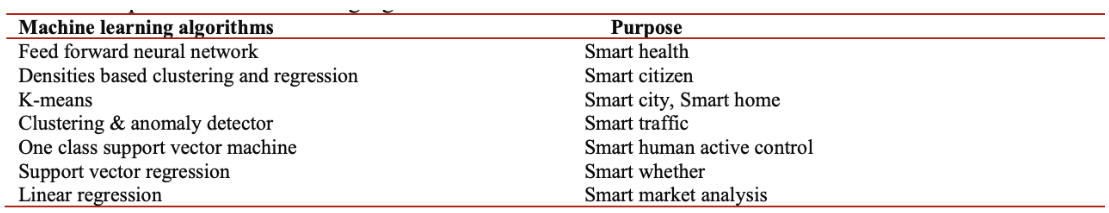
- Label Bias and Environmental Datashift
- Suitable Algorithm Selection
- Reward Hacking

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* [4]. Each model will be trained and evaluated using data which 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 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 generalize.

2.2 Suitable Algorithm Selection

As can be seen 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 of the model type. Accuracy, precision and recall are three common metrics we can use to evaluate a model. However, depending on certain 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 models 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 [13].

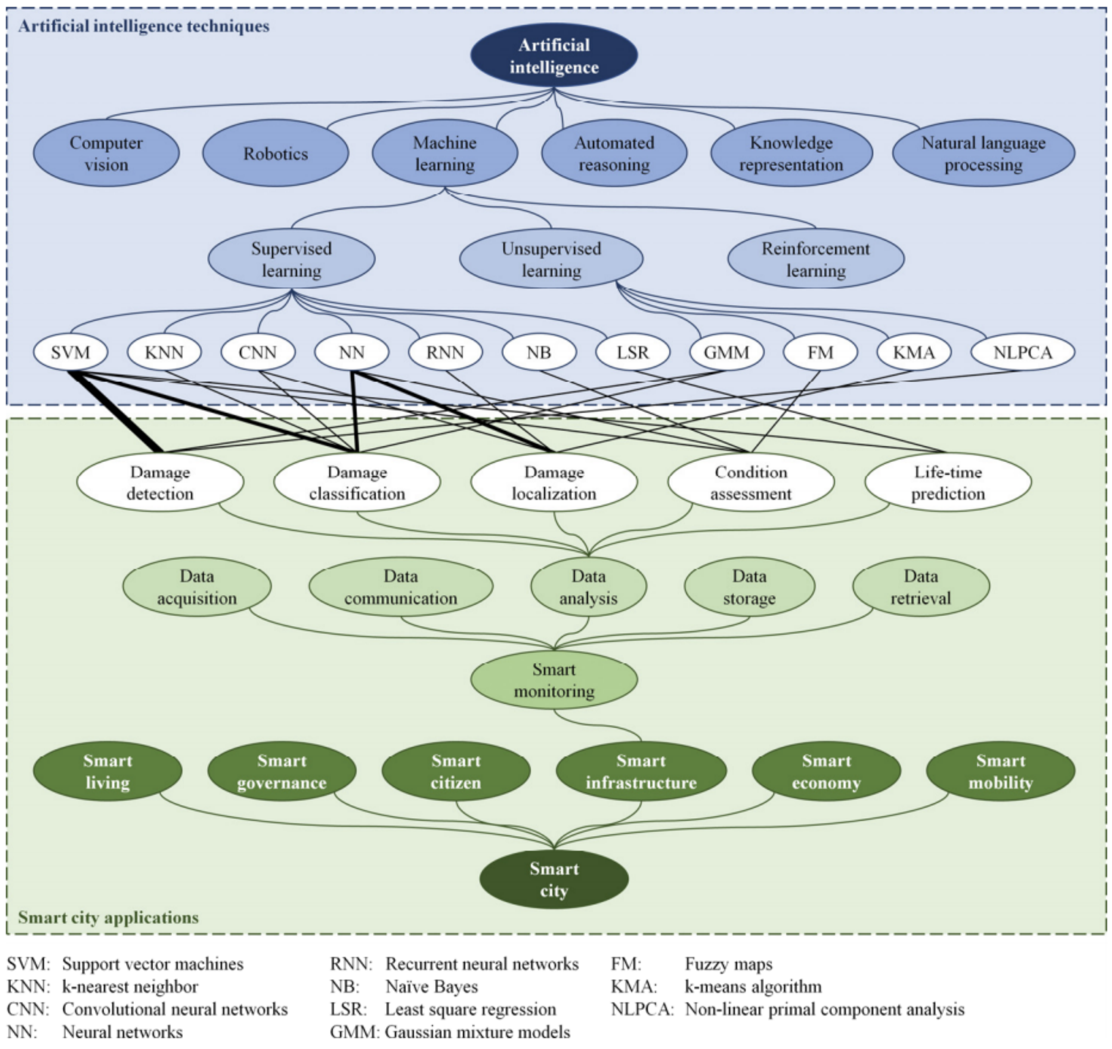


Figure 2.2: Available ML algorithms for smart monitoring [3]

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 full data set, while the validation and test sets account for 20% each.

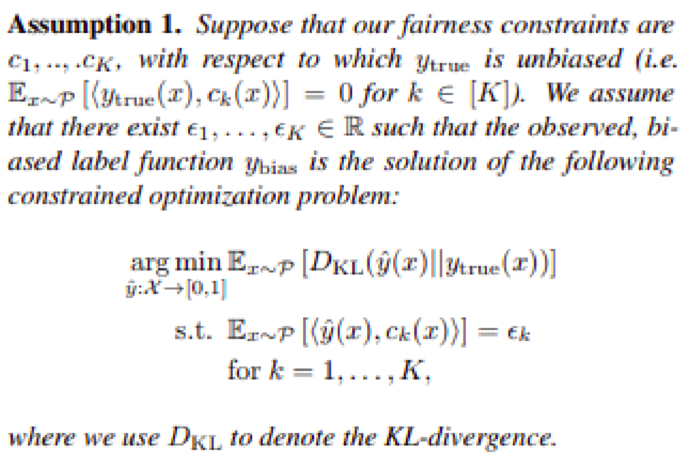
2.3 Reward Hacking

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 as well as potential solutions.

Label Bias and Environmental Datashift

Bias is the result of inadequate data where a certain group or class is favoured over another/others hence creating an overrepresentation [4] [11]. ML models trained using such datasets will acquire these underlying biases hence making incorrect predictions.

The following mathematical framework can be used as a representation to undestand bias in data [4]



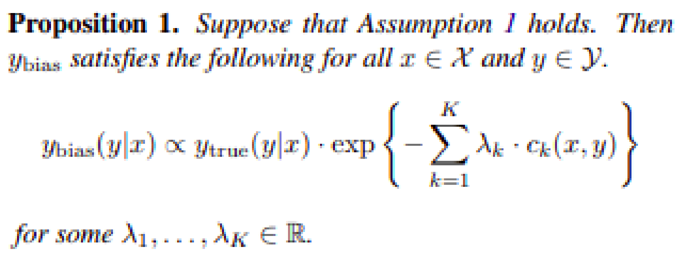
Assumption 1. *Suppose that our fairness constraints are $c_1, .., .c_K$, with respect to which y_{true} is unbiased (i.e. $\mathbb{E}_{x\sim\mathcal{P}}[\langle y_{\text{true}}(x), c_k(x)\rangle] = 0$ for $k \in [K]$). We assume that there exist $\epsilon_1, \ldots, \epsilon_K \in \mathbb{R}$ such that the observed, biased label function y_{bias} is the solution of the following constrained optimization problem:*

$$\begin{aligned} \underset{\hat{y}:\mathcal{X}\to[0,1]}{\arg\min}\ & \mathbb{E}_{x\sim\mathcal{P}}\left[D_{\text{KL}}(\hat{y}(x)\|y_{\text{true}}(x))\right] \\ \text{s.t. }\ & \mathbb{E}_{x\sim\mathcal{P}}\left[\langle \hat{y}(x), c_k(x)\rangle\right] = \epsilon_k \\ & \text{for } k = 1, \ldots, K, \end{aligned}$$

where we use D_{KL} to denote the KL-divergence.

Figure 3.1: Bias Assumption [4]

In figure 3.1, the assumption is that y_{bias} is the label which is closest to y_{true} and achieves the same amount of bias. In cases where data has been manually manipulated by humans, either consciously or subconsciously, this is deemed to be a reasonable assumption. The contiguity to y_{true} is given by the KL-divergance, which is used to establish the notion of accurate labeling. The Proposition in figure 3.2 is derived from the KL-divergence. (For complete proof of proposition, see [4])



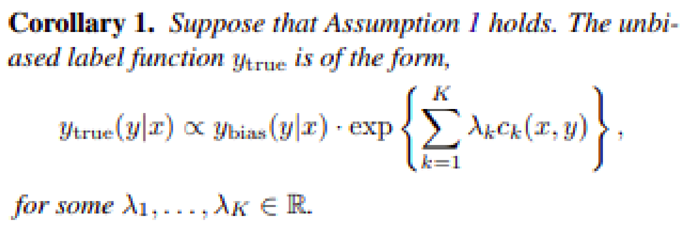
Proposition 1. *Suppose that Assumption 1 holds. Then y_{bias} satisfies the following for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$.*

$$y_{\text{bias}}(y|x) \propto y_{\text{true}}(y|x) \cdot \exp\left\{-\sum_{k=1}^{K} \lambda_k \cdot c_k(x,y)\right\}$$

for some $\lambda_1, \ldots, \lambda_K \in \mathbb{R}$.

Figure 3.2: Bias Proposition [4]

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. *Suppose that Assumption 1 holds. The unbiased label function y_{true} is of the form,*

$$y_{\text{true}}(y|x) \propto y_{\text{bias}}(y|x) \cdot \exp\left\{\sum_{k=1}^{K} \lambda_k c_k(x,y)\right\},$$

for some $\lambda_1, \ldots, \lambda_K \in \mathbb{R}$.

Figure 3.3: Bias Corollary [4]

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 [11]. 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.

3.1 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. Each cycle of data reading contains the following measurements:

3.2 Results

Consider noise in dataset due to uneccassry labels

Datashift: The issue is that modelers typically assume that training data is representative of the target population or environment where the model will be deployed [11]

3.3 Discussion

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 in turn is dependant on the amount and type of data available. Another key factor is the use case of the model and its intended outcomes. Generally model parameters are curated for specific applications and will differ to other use cases. Therefore, it is important to make use of inductive bias [11] to when developing reliable models.

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. Each cycle of data reading contains the following measurements:

Table 4.1: 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. Device failure is to be classified when the time remaining to device failure is less than one hour.

This dataset has been split into two datasets for training and testing respectively. The training dataset will compromise of all data collected from gadget IDs 1-4, leaving data from gadgets 5 and 6 for the test set. This will ensure the trained models are tested on completely new data.

For more information on the dataset and use case, please see [] https://github.com/Unikie/predictive-maintenance-tutorial

4.2 Algorithms

The following section shortlists and describes algorithms which 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 thge 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 vecotrs(closest points), i.e. maximise the minimum margin.

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

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

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,

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

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:

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

A linear kernel will be used in this experiment.

4.2.2 k-Nearest Neighbours

k-Nearst Neighbours (k-NN) is a simple supervised machine learning algorithm used in both classification and regression problems. This approach classifies objects based on the compuatational 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. The choice of k will affect the algorithm's performance where a value too small would create higher variance hence 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.

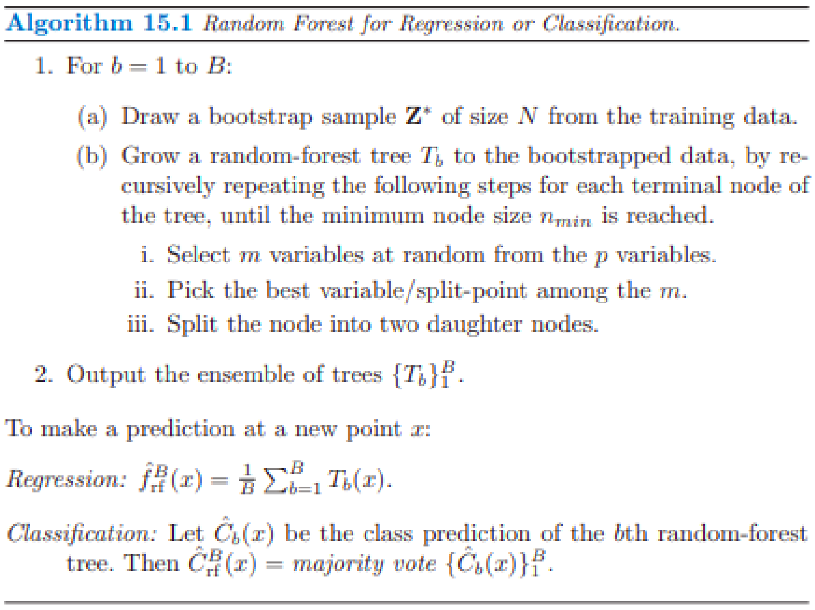
$$d(x_q, x_i) = \sqrt{\sum_{i=1}^{m} (x_q - x_i)^2} \tag{4.5}$$

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

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 models 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 which creates multiple random decision trees and combines their respective votes (classification) or averages (regression) to improve prediction accuracy and fitting.



Algorithm 15.1 *Random Forest for Regression or Classification.*

```
1. For $b = 1$ to $B$:
    (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.
    (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by re-
        cursively repeating the following steps for each terminal node of
        the tree, until the minimum node size $n_{min}$ is reached.
          i. Select $m$ variables at random from the $p$ variables.
         ii. Pick the best variable/split-point among the $m$.
        iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.
```

To make a prediction at a new point x:

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

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\rm rf}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Add more stuff about reliability/performance factors

4.2.4 Neural Networks

Some filler text for the time being.

4.3 Results

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

- 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)

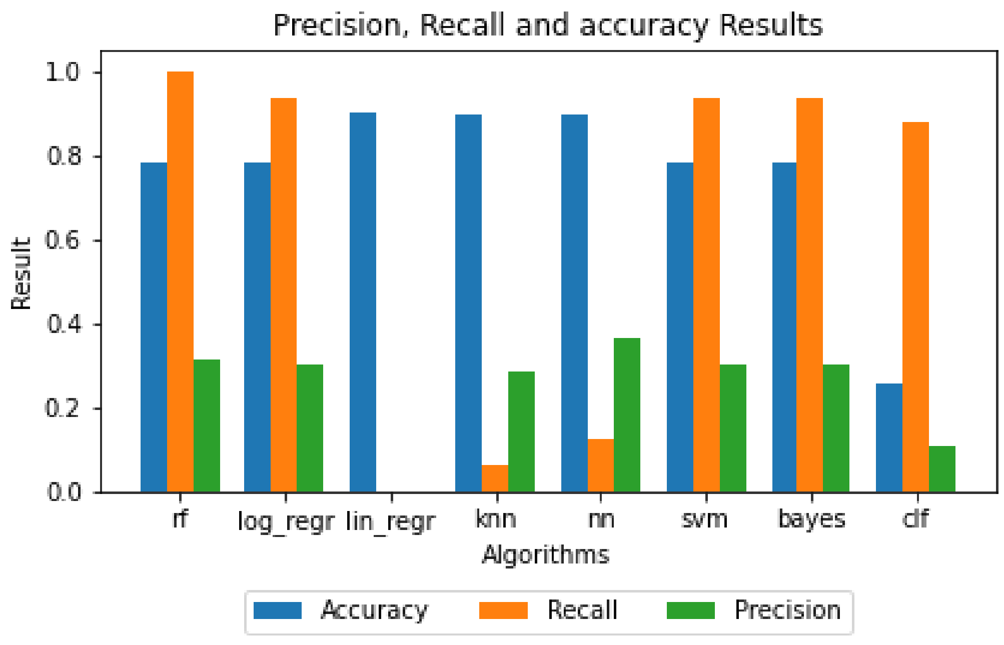


Figure 4.1: Precision, Recall and Accuracy Scores

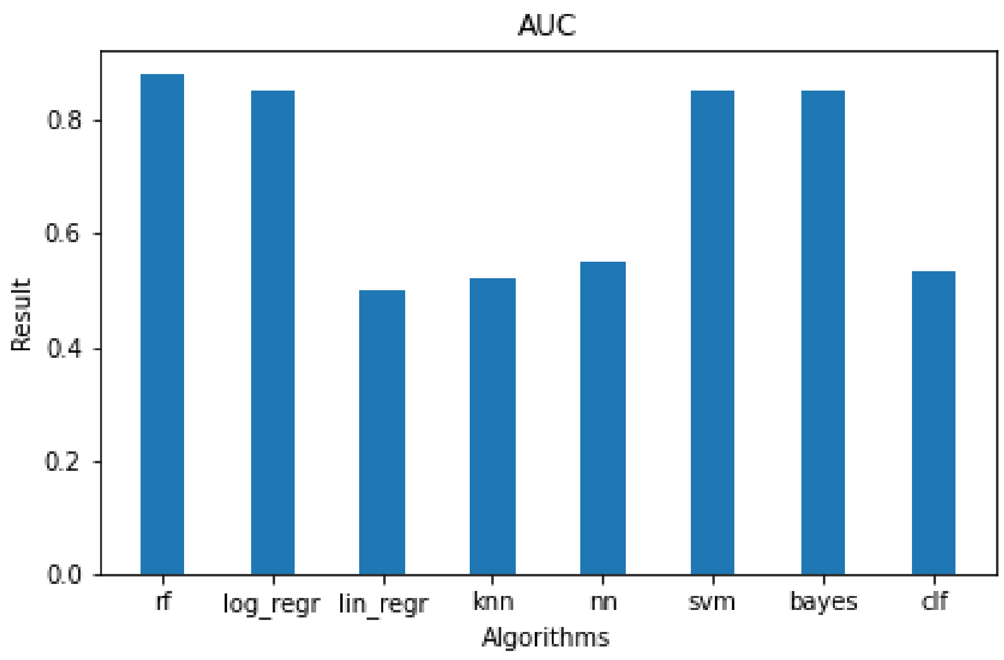


Figure 4.2: Area Under the Curve

Table 4.2: 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 4.2, highlighted are the highest comparison scores between tested algorithms. The ensemble method Random Forests is evidently the most suited method to this predictive maintenace 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.

Table 4.3: 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. A False Negative (FN) case occurs when the model predicts a non-failure but in reality a failure has occured. This would result in large financial penalties for the company due to downtime. Therefore, the highest amount of TP cases and lowest amount of FN cases are desired.

4.4 Further Research

A seperate study compares existing resgression 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'* [5].

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 occurs 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 accuracy of the tested algorithms.

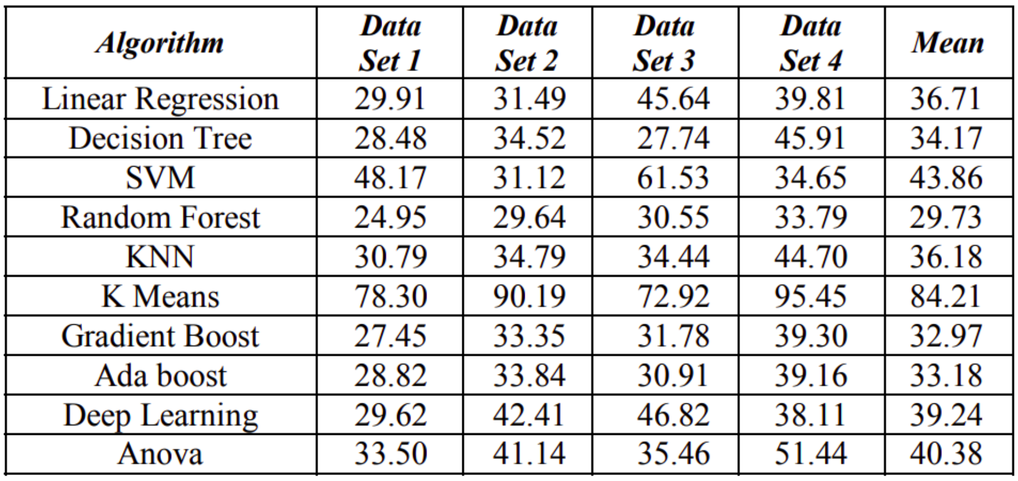
Listed below are machine learning algorithms used to predict RUL in this study:

- 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 4.4: RUL Algorithms Root Mean Square Error values [5]



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 4.3: RUL Algorithms Root Mean Square Error Plots per data set [5]

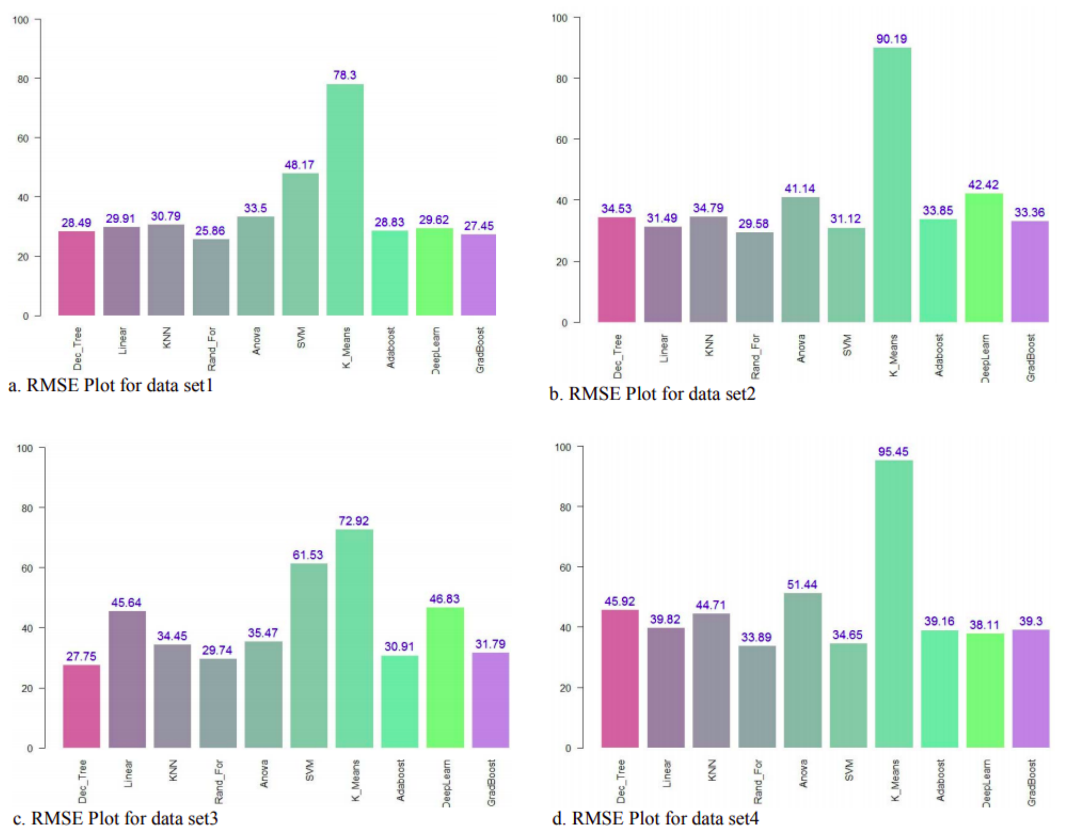


Table 4.4 and Figure 4.3 displays the RMSE values of all tested algorithms across all four datasets. As can clearly be seen, the complexity of this problem results in fluctuation of evaluation scores unlike our previous classification experiment. RMSE is a measure of the concentration of data around 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 experiement, it is clear that enseble methods perform the best, specifically, Random Forest. It should also be noted that SVM outperformed knn in the classification experiement but performed worse in the RUL regression study.

add another study

4.5 Recommendations

Above sections highlight the importance of correct model spcifications. Previous assumptions and expectations may not always be correct.

Use ensemble methods and hyperparameter tuning.

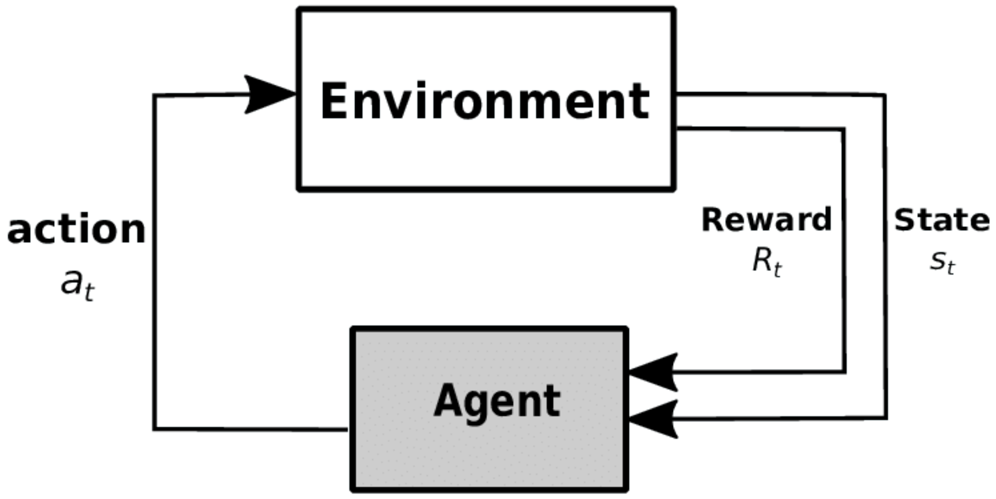
It is also important to select appropriate evaluation metrics based on the model type and application. Assign a weight or cost to each output of the confusion matrix

Research specific use case scenario to find suitable algorithms through proven research

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 ineteract and manipulate the environment, state, S_t, and a reward fucntion, 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 recieved.

Figure 5.1: Reinforcement Learning and the Environment [6]



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 slso succeptible 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. This 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 [14].

An example of this RL scenario [14], 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:

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

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 [15] aims to maximise the amount of vehicles through intersections while minimising 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 certain queue within an intersection, or it may completely 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 [16]. Or an agent controlling a video game may choose to maximise the collection of points rather than completing the goal/mission [16]. 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 [17].

In the following two sections we identify and discuss causes of reward hacking and potential solutions as presented in studies caried out by researchers primarily from Google [12] [18].

5.2 Causes of Unreliability

The following sections describe some common causes of Reward hacking. There is no one 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 objectivive 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 [12]. The example provided in the previous section is a great example of this. The standard deviation was intended to equalise the importance of all queues. But 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 interelationship betwen an objective function and its means or factors to successfully accomplishing a task is too high, that relationship will not hold under heavy optimisation [12]. Continuing on with our TMS example, if the success of the operation were only based on throuput of vehicles across an intersection, the agent may 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 for keep the queue count at zero.

5.2.3 Wireheading & Reward Function Tampering

Wireheading is ability of an intelligent agent to modify its reward sensors to ensure it always recieves the maximum reward [19] [12]. In short, Wireheading occurs if the agent is no longer optimisising the the objective function, but rather 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 [18]. 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 [18]. Concern arises in cases where humans have influence in the reward process (implicityly 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 reward function is manipulated by the agent itself, Partially Observed Goals is the result of the designer's inadequate reward function design. In turn the inadequate design is the result of partial observations of the deployment environment and/or features which are difficult or impossible to measure. Therefore assumptions have to be made in an attempt to develop a reward function [12].

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 [12]. 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 sesnor 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 simplest yet most tedious solution to reward hacking is careful engineering [12]. Performing formal verification or thorough practical testing is an easy way to iron out issues which 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 which is significantly larger than optimising the function in the desired manner [12].

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 objective function, ensuring that an invulnerability in one reward function does not impact the whole system [12]. 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 which 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 which can cause unreliabilties in the reward function. If the reward function is designed only considering the training environment and its respective features, the behavior of the agent in differing 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 to recover r

Where:

- R is a sapce 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

For a detailed mathematical explanation of Inverse Reward Design see, [16]

5.3.2.4 History Based Rewards

The following claim has been made in [18]:

> *"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 which are competent in following 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

Add content here

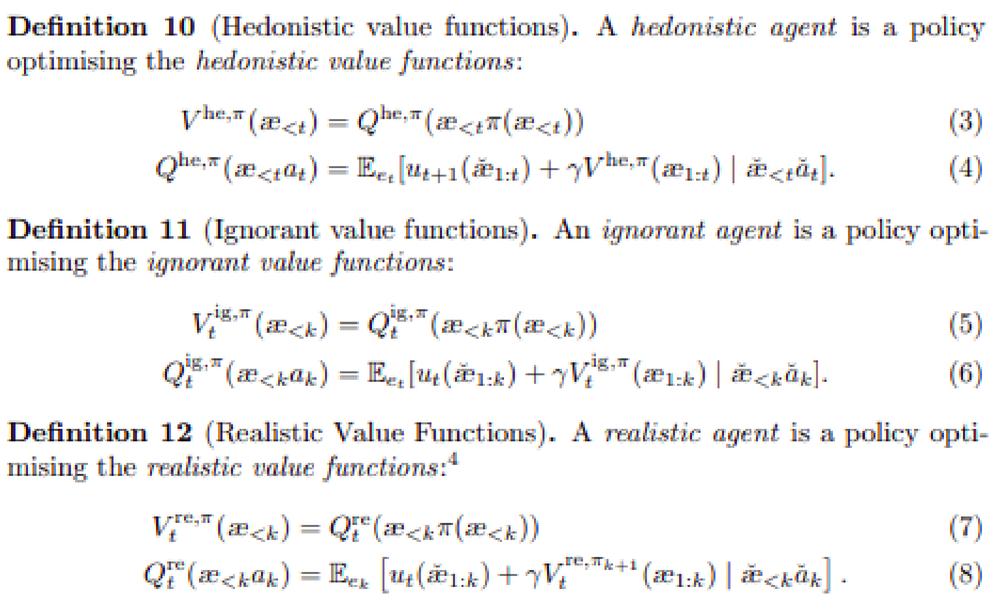
5.3.3 Model Lookahead

A viable solution to prevent the model/agent from replacing its reward function (Reward Tampering) is to implelement Model Lookahead. A model based RL system uses a model to develop a strategy for future actions by assessing the states a seiries 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 [12]. The agent will not be able benefit from hacking as these exploitations will most likely not reach the anticipated states. To represent Mathematically:

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

This notion continues in a similar study (see for complete mathematical break-down) [7] 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 [7]


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

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

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

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

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

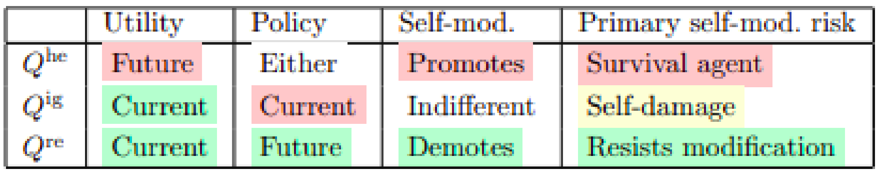
$$Q_t^{\mathrm{ig},\pi}(\text{æ}_{<k}a_k) = \mathbb{E}_{e_t}[u_t(\check{\text{æ}}_{1:k}) + \gamma V_t^{\mathrm{ig},\pi}(\text{æ}_{1:k}) \mid \check{\text{æ}}_{<k}\check{a}_k]. \tag{6}$$

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

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

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

Table 5.1: Self-modification value functions [7]



	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 describe as adversarial [12]. 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 tecnhinque 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 Trip Wires

Trip wires is a technique akin to a security system. The main goal of trip wires is to alert the human that the agent is attempting to exploit the reward function [12]. 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 risk of harmful actions but also act a measurable quantity or diagnostic which can be used to further improve the objective function.

5.3.6 Algorithmic Approaches

Researchers in [20] 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. 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 immidiate reward. For a detailed breakdown of this approach, it is recommended to visit [20]

For further reading, algorithmic approaches for wireheading and reward function tampering are discussed in [19] and [18] respectively.

Conclusion

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

[5] 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.

[6] 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.

[7] 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.

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

[9] R. Moutafis, “How bad facial recognition software gets black people arrested..” towardsdatascience, 2020.

[10] 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.

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

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

[13] 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.

[14] 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.

[15] 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.

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

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

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

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

[20] 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.