Rule Based Incident Response for Vehicle's Safety and Situational Awareness

Author: Mark Burnitt

Supervisor: Neetesh Saxena

Moderator: Stuart Allen

Module: CM2303

Credits: 40

Abstract

The in-vehicle CAN network was conceived and released to market 35 years go. In that time, there has never been a security patch or changes to the specification to address the security issues it has and as the cyber threat landscape has evolved and vehicles become more technological and connected, cyber-attacks against the in-vehicle network are now a growing threat. This work investigates the possibility of using a combination rule-based and machine learning based classification and detection system for attacks against the CAN bus. This is done by adding extra attributes to the raw CAN data, then feeding it through the processes sequentially. A valid ID rule and a time interval rule were used for the rule-based stage. A small number of machine learning models were tested for both the detection and classification stages of the proposed system, with the best performance being the AdaBoost algorithm for detection and the Gradient Boosting algorithm for classification, with the combination using a minimal amount of memory and being able to perform within the real time constraint. The performance of the other algorithms compared to these two were variable across all attributes.

Acknowledgements

I would like to acknowledge my project supervisor for this work, Neetesh Saxena, for his continued guidance and feedback about the work undertaken as part of this project.

Table of Contents

1 Introduction

1.1 Aims of this Project

This project aims to improve on the situational awareness of the in-vehicle network, with the intent of being having additional or future work be able to leverage the enhancements provided by this project for the benefit of the network and its users. It will do this by being able to analyse the traffic being passed across the vehicle CAN bus(es) and using only this to be able to detect when the vehicle is the target of a cyber-attack, as well as being able to identify the type of attack that is taking place, in a time that is relevant with the real-time speed of the CAN network.

1.2 Approach and Scope

The proposed system in this project will carry out its intended aim by using a rule-based system to improve the situational awareness of the vehicle. This will work alongside machine learning techniques, with the intent that the system will be able to recognize attacks it is both familiar and unfamiliar with, with a high detection rate and low false positive rate. The same applies for the ability to classify the type of attack in the traffic. It will only be tested using the data from the dataset that is available, and as such will only consider attacks that are inside the attack model that this dataset represents. However, it is the intent of the project to be able to test how the system responds to attacks that it has not been trained on. It is also beyond the scope of this work to consider taking any actions from the attacks that are identified and classified, merely to stop at the identification and classification stages. The system will also only be designed for CAN 2.0A frames, not any of the further extensions or variations that can be present, or other types of network that may be present in the in-vehicle network.

1.3 Intended Targets

The system is targeted or being able to be deployed in any kind of vehicle that uses the CAN 2.0A specification. It could either be fitted into dedicated ECU(s) or added as an aftermarket product through the OBD-II port. It is intended to be deployed by those who have specialist technical knowledge, as it may require some tweaks between vehicles due to the fact that the IDs used between networks are non-standard and therefore vary from manufacturer to manufacturer.

1.4 Reasons for Work

The reason that this work is being carried out is that the CAN network is not inherently secure by design and that none of the existing approaches proposed in research have yet been adopted into practice. Traditional security techniques applied to more conventional networks cannot be used for reasons such as the low data rates used in the CAN network (often less than 1Mbps) or being too slow, so a different approach is needed. There are also issues in the design of the CAN network itself, being a broadcast system with no authentication or freshness checking for messages. As such, attackers are leveraging this to be able to successfully perform attacks against the CAN network to a variety of results, from disabling components to being able to remotely control the vehicle. Therefore, a system is needed to detect when attacks against the network occur, to provide it with some level of defence against malicious actors.

It is also often the case that the existing solutions proposed in research have issues that hamper them in one area or another. As a lot of these are neural network based, trying to improve the performance of the system created requires the adding of more neurons or layers, which lowers the speed of prediction and increases the size required for the network in memory. They can also be limited by the type of attacks that they can cover, as some methods rely on characteristics that are not present in every kind of CAN message or may have issues when attempting to be translated to an ECU spec environment, rather than a traditional computer.

2 Background

2.1 Overview

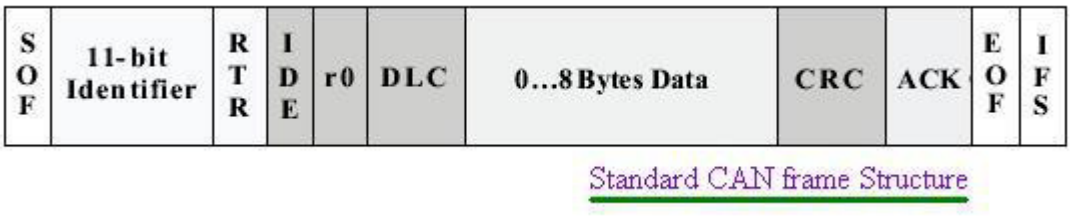
This section covers the necessary background information to sufficiently understand the rest of the work. It will cover the basics of the in-vehicle network, how communication is structured on it, the reasons for wanting to keep the security of the network intact and some of the challenges faced when doing so. This will be done by examining some of the existing work in the field and identifying the motivations behind it, as well as where there are problems that have not yet been addressed.

2.2 The Controller Area Network (CAN).

2.1.1 The Basics

Prior to the invention of the CAN bus and its release to the public by Bosch in 1986, the network in vehicles was created with a lot of point-to-point wiring between the ECU units. This was both expensive and difficult to repair should any faults occur, a problem that the CAN bus does not have. Rather than have the point-to-point connections, the ECUs are instead all wired into a single CAN network (though this can consist of multiple physical buses, with gateways in-between). This both reduces the cost as it requires far less cabling, and improves the fault tolerance of the network, as it is highly unlikely for the entire bus to fail, merely single ECUs, barring extreme circumstances. The ECUs utilize the bus by sending packets or "frames" onto it, which have the structure detailed below.

Fig 1. The structure of a CAN 2.0 frame.



- SOF: Start of Frame (1-bit) – Signals the frame beginning
- (11-bit) ID – Used for bus arbitration, with lower numbers carrying the maximum priority (minimum of 0)
- RTR: Remote Transmission Request (1-bit) – This will be set to 1 whenever an ECU needs data from another ECU. Though all ECUs receive it, the ID will determine which ECU replies.
- IDE: IDentifier Extension (1-bit) – This signals that this is a standard CAN frame, not one of the extended formats.
- R0 (1-bit): Not currently in use, but reserved
- DLC: Data Length Code (4-bits) – Tells how many bytes to expect as part of the actual data section of the frame
- Data (0-64-bits) – Whatever values and other information the frame needs to carry, to a maximum of 8 bytes.

- CRC: Cyclic Redundancy Check (15-bits) – One of the error detection methods, used like a checksum for the rest of the frame.
- ACK: ACKnowlegement (3-bits) – Used to signify whether the intended ECU received the message correctly, or to report an error in transmission.
- EOF: End Of Frame (7-bits) – Marks the end of the frame.
- IFS: Inter-Frame Space (3-bits) – Used to make the ECU wait while it moves a correctly received message into the message buffer.

This is the original structure of the CAN frame, through it has been updated several times since then, most notably including CAN 2.0B, which has 29-bit identifiers (though is backwards compatible with 2.0A) and CAN-FD which can have up to 64 bytes of data and much faster transmission speeds. Other systems have also been developed as alternatives, including the Local Interconnect Network (LIN) which is used as a replacement for CAN in non-safety critical applications, due to its lower cost and FlexRay, which was designed to be faster and more reliable than CAN, but has seen limited adoption.

2.1.2 Why the CAN network needs securing.

The CAN bus neatly avoids the issues of complex point to point ECU connections, both reducing the cost to manufacturers as well as the complexity of repairs. However, the system was first released in 1986, where the idea of performing cyber-attacks against a car was remote at best. In more recent times however, as the cyber threat environment has evolved, and cars ever more packed with technology, the car has become more and more vulnerable to attacks. This included attacks from both physical vectors (the OBD-II port and compromised ECUs) as well as remote vectors. For example, Miller and Valasek [1] exploited an unmodified Jeep Cherokee and were able to remotely control the vehicles steering, brakes and engine. Had they had malicious intent, they would have had the potential to to cause an immense amount of damage to life and property before someone realized what they were doing and were able to develop the appropriate patch. A more comprehensive chart displaying the attack vectors is available in Bozdal et al. [2].

As there has not been update regarding security to the CAN bus since its inception, the security of the network has only decreased as more and more ways to access the network are found. This is because the CAN network does not implement any forms of encryption or authentication, and is a broadcast system, meaning every ECU can both transmit and receive to every other ECU. If nothing is done to rectify this, attacks like that of Miller and Valasek will become more common and move from the domain of security research to that of the cyber-criminal. Upstream Security's Automotive Cybersecurity Report for 2020[3], reveals that the number of cyber-attacks against vehicles has increased by 94% Year-on-Year since 2016, with 57% of the attacks in 2019 being made by "black hat" hackers.

2.1.3 Challenges Faced in Securing the CAN network

While it is known that the CAN bus is not entirely secure, there exist several challenges to trying to make it so. The CAN bus only supports very low data rates, sometimes as low as 250kbps for a variant of CAN 2.0A (ISO standard 11519, though subsequent revisions increase this), so traditional techniques with large overheads like software based public key cryptography are unsuitable to be directly translated over to the CAN network. That

is not to say that possible solutions do not exist, as both Siddiqui et al. [4] and Jukul and Čupera [5] propose encryption approaches in their respective papers. The former uses extra hardware inside the system to implement AES-128 symmetric key cryptography, the latter the Tiny Encryption Algorithm, a kind of Feistel Cipher. Authentication protocols have also been proposed, such as the LeiA, the work of Radu and Garcia [6], which applies the concept of Message Authentication Codes to the CAN network. However, even if these systems were able to be commercialized and implemented, the problem does not stop there. Relying on a single technique to keep the bus secure would be naïve at best, so there is no harm done by exploring other techniques that could potentially be used in parallel.

There are also challenges that arise due to the way the network is designed and is used. One of these is that the CAN network is by design, a broadcast system. This is unable to be changed, as this is a fundamental part of the CAN architecture, but it when combined with the fact that the network has no authentication either, its problem is compounded. It makes it much easier to conduct every single kind of attack against the network, as the attacker has a much larger surface to attack and compromises in smaller systems (such as the TPMS) can affect other ECUs that may be more critical to vehicle operation. Another issue with the fundamental design of the CAN network is that way it handles bus arbitration. Each CAN frame has a 11- or 29-bit identifier that immediately follows the SOF bit. The lower that this number is, the higher priority that frame has on the bus. This can be leveraged by attackers to conduct DoS attacks and since as this is a fundamental part of the protocol, it cannot be patched or changed. Finally, there is the need that the network operate on a real time basis. Even a system that could flawlessly secure communications or detect attacks on the network is useless if it cannot keep up with the speed of messages across the bus and give responses within a very short period of time. This is another reason why a lot of the traditional security techniques are not permissible in the CAN environment, as they are just too slow.

2.2 Prior Work.

2.2.1 Indicators of Compromise

Like cyber-attacks against traditional networks or machines, it is extremely difficult, if not impossible for the activities of the attacker to be invisible. In the traditional environment, there may be things like files being created, accessed, or modified, or new and persistent processes that run even when they are not supposed to. In the CAN system, there is not these indicators, as the type of data accessible is much more low level. However, each attack still leaves traces on the network that can be used by an IDS to detect when and what has taken place, without any additional information.

These indicators can take a number of different forms. The most obvious of these is of course, the physical effect of the attack, but this is not able to be used as part of the system, as this is both difficult to define computationally and somewhat redundant to prevent the consequences by detecting those same consequences. More useful indicators are left by each kind of attack, which can be used to identify and classify them as per the aims of this proposed system. For example, a DoS attack that uses the method of flooding the bus with maximum priority messages, it can be observed that a large concentration of the same (or very similar) high priority CAN IDs are on the bus in a short space of time. This is unlikely to happen during normal network operation and is almost impossible for a

legitimate number of messages to be generated to prevent the use of the bus. The ID can also be used to perform a check for new additions to the network, if an ID is observed that has never been seen before (though this requires the attacker to be actively broadcasting, or have done so at least once), as it is logically impossible to modify the network while it is under operation under normal circumstances (as it would be sealed inside the body of a moving car), and therefore can correlate to an attack.

For message injection type attacks, several things can be observed, depending on what kind of attack that may be underway. A spoofing type of attack, that aims to generate erroneous values and have this displayed to the driver (e.g. making the RPM gauge read higher than normal), will always use the legitimate ID of the targeted component but will be putting messages onto the bus much faster than normal, to have them actually received and processed by the target ECU. It is also possible to examine the data content of the CAN frame, even if this cannot actually be decoded into a coherent value. The way that this can be done is proposed by Müter and Asaj [7] who use the entropy (the randomness of the data) to be able to identify when someone has changed the data contents, as this results in a noticeable spike in entropy. For fuzzy attacks that generate random message IDs and data, this entropy-based method should be able to function as intended, although it cannot classify the type of attack in progress as it is an anomaly detector. If the contents were able to be understood, there would be the additional option of adding a check to see whether the value was actually feasible or not (instantaneous high acceleration, engine RPM higher than maximum etc), though this requires the data to be able to be understood and may not be universal across all vehicles. An example of such an approach that uses deep learning was presented by Li [8] at his talk at the DEFCON conference.

2.2.2 Existing Solutions

As the security issues with the CAN bus are not a new problem, a number of proposed solutions exist in research papers, and there are a few systems that are commercially available that put into practice a rule-based approach, similar to traditional ID(P)S for a network. These generally fall into one of four categories, based on the approach used.

2.2.2.1 Physical Characteristics

The first type uses physical characteristics of the system to build a profile of the particular CAN network, and then compares message traffic against this after it has been constructed. An example of this kind of system can be found in the work of Cho and Shin [9], who use the clock skew of the individual ECU as a method of digital fingerprinting. While this method was highly accurate, and resistant to attacking itself (as this would need to physically heat or cool the ECUs), it does not fundamentally function for an ECU that does not have its traffic sent periodically, as the clock skew cannot be measured in this case. It would also require multiple ECUs to be installed into the vehicle with this system running, to prevent the attacker simply compromising the IDS ECU before conducting a more typical attack.

2.2.2.2 Strictly Rule Based

The second approach is to adopt a strictly rule based system, which is an approach used by some IDS systems for traditional networks. This would simply compare the CAN traffic against its profile of threats and then decide on whether the given frame was malicious or not. This has the advantage of being easy to implement and extremely fast

(an important factor in real-time networks like CAN), but also has a number of disadvantages. The most obvious of these being that if the attacker is using a zero-day exploit, or the system has not been configured correctly, attacks would be able to pass through undetected. Despite this, there is a commercial approach by Arliou, Sentinel-CAN [10], that is already on the market. Research approaches also exist, such as the work of Marchetti and Stabili [11], who analyse the sequence of IDs that pass across the bus, and Song et al. [12], who do the same thing but focus on the time intervals between messages. These are examples of simple, lightweight systems that use rules relating to a single attribute to function, but this comes with the limitations of not being able to detect certain kinds of attack at all (mainly being limited to message injection).

2.2.2.3 Strictly Machine Learning Based

Next, there is the approach of applying a machine learning algorithm to the problem. Various approaches have been tried, using a large variety of types of algorithm, but there are a number that see high performance or hold specific advantages that the others do not. These are the Generative Adversarial Network (GAN), used by Seo et al [13] with very promising results, despite its ability to suffer confusion with events such as ECU failure. Song et al [14] use a Deep Convolutional Neural Network (DCNN) instead. The DCNN was slightly more accurate than the GAN approach, but must be trained in a supervised manner, meaning it struggles to identify unknown classes of attacks. Both Loukas et al [15] and Qin et al [16] apply LSTM approaches. These are the traditional type of network employed when there is data over time to analyse. While of lesser accuracy than the GAN or DCNN approaches, they offer a different way of modelling the problem, which may be more likely to detect certain classes of attack, or more able to analyse certain elements of the data. Loukas's approach also offers the advantage of offloaded computing, though only while the car is connected to a network, making it able to be much more computationally complex for the same response time.

2.2.2.4 Hybrid Approaches

Finally, there is the combination approach. This combines one or more of the above, in an effort to make the system more than the sum of its parts. Zhang et al [17] used a combination rule and machine learning based system to achieve near flawless accuracy against their chosen attack model, with very low processing times, though they did not consider how feasible this approach would be to implement with all the constraints of ECU hardware, and do not cover any spoofing type attacks in their attack model.

Table 1. Summary of existing IDS type systems for the vehicle network.

Reference	Approach	Research/Commercial	Results	Limitations
Seo et al [13]	Machine Learning– GAN	Research	Average Detection Accuracy of 98%	The system has the possibility to be confused by ECU errors, misidentifying them as malicious activity.
Song et al [14]	Machine Learning– DCNN	Research	Around 99%	The system learns in a supervised manner, which is to say that the system would struggle to identify attacks that it has not been trained on, as it would not know what features to examine.
Kang and Kang [18]	Machine Learning– DNN	Research	Overall Accuracy of 97.8%	With this system as described, to try and improve on the reported stats would be difficult, as the more layers are added, the slower the system will become.
Arliou [10]	Rule Based.	Commercial	N/A	As Sentinel-CAN is a strictly rule based system, similar to systems like Snort for traditional networks, its effectiveness is solely dependent on the defined rules. Attacks of an unknown signature would bypass the system if they were within the normal “communication profile”.
Young et al. [19]	Message Frequency Based	Research	Detection Accuracy of 100%, for message injection only.	This accuracy still has a false positive error similar (if slightly lower) than the machine learning based approaches and only detects attacks that require message injection.
Zhang et al [17]	Combination Rule Based and Machine Learning (DNN)	Research	Detection Rate 99.9%	System does not consider attacks outside of its attack model and may not be feasible to implement within the constraints of ECU hardware.
Loukas et al [15]	Machine Learning – LSTM	Research	Accuracy 87%	The advantage of this method of being able to offload the computation to cloud servers only works in connected environments. The attacker can deliberately attack the vehicles connection to force the system into a lesser state of effectiveness.
Song et al [12]	ID Pattern Based	Research	100% for Bad Injection and Random Injection 20-40% for Replay Attacks	This approach only considers message injection attacks but does so in a more realistic way. The system also struggles against replay and single frame random injection.
Qin et al [16]	Machine Learning– LSTM	Research	Over 90% Accuracy	System considers tampering with existing messages on a per-ID basis and as such, the scaling issues present are rather high.

				System does not cover the possibility of attacks that do not need tamper with the payload of messages.
Marchetti and Stabili [11]	Message Time Intervals	Research	Reported 100%	System tested against extremely small samples that are not really representative of normal CAN data.
Cho and Shin [9]	Clock Skew	Research	Close to 100%	The system requires dedicated ECUs to be installed (more than one to prevent the attacker just compromising the IDS ECU) The system only works for messages whose traffic is periodic and cannot detect attacks when the ECU does not have periodic traffic.
Markovitz and Wool [20]	Message Fields	Research	"Very Encouraging"	System was tested using only 10 ECUs in a simulated environment, so may not be feasible for the real system, and only detects anomalies, it does not classify them.

2.2.3 Performance Assessment Methods

2.2.3.1 Precision, Recall and F1 Score

Precision is a measure of how accurate the system is, given by the number of true positives identified, divided by the sum of the true and false positives. Especially for identifying the malicious and benign traffic because the vast majority of the traffic is attack free, a relatively high precision (and perfect recall) could be obtained by just classifying everything as an attack. This must be avoided, as a low number of false positives is important in this particular environment, as there are very limited resources available, so they should not be wasted on chasing perfectly normal traffic. A Precision that is extremely close to, if not 1 would be an ideal outcome for these experiments. Precision has been chosen as a method of assessing the system over accuracy because of the imbalanced nature of the dataset (and the problem at large). As mentioned earlier, the vast majority of packets are attack free, so very high accuracy could be obtained by just classifying everything as benign, which would defeat the whole idea of the system. It is more meaningful to consider how many of the attacks are identified, as well as how many false positives are being generated, as these are not desirable in the safety critical CAN environment.

Formula for calculating Precision

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

Recall is the measure of how many true positives were identified, out of all the true positives and the false negatives. This should also be extremely close to, if not 1 in an ideal system, as every attack that manages to slip through the detection system could potentially have some drastic consequences for the occupants of the vehicle. Of the two, it is preferable for this case to have a higher recall, even if it means marginally lower precision, as the waste of system resources is preferable to the potential consequences of a compromise. Recall is a superior method to plain accuracy and is being used for much the same reason that precision is. It is a

more precise measure, suitable for the imbalanced dataset in use, and considers a specific class of error, in this case the false negative (which could be much worse than the false positive).

Formula for calculating Recall

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

The F1 score is the harmonic mean of the above two. This allows us to collate the two above statistics into a single one, that represents how good the balance between the two component values is. This allows us to identify (if a score of 1 cannot be achieved) the point that is the best trade-off between the precision and recall values (the highest F1 score). The F1 score is important because it combines and relates precision and recall together, allowing a single number to represent the overall performance of the system, which can then be very easily compared to existing work or taken and turned into appropriate charts to display the differences when using variations in the approach and how much this impacts the system as a whole.

Formula for calculating F1 Score (of a two-variable system)

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

2.2.3.2 Confusion Matrices

Confusion Matrices are a fast, extensible, and visual way of identifying exactly which attributes are being misclassified. For a given (binary) classification problem, such as determining whether the given CAN frame is malicious or not, there are two possible correct values, 'Yes' or 'No'. If these are aggregated for every frame tested, it can be summarized by a confusion matrix that would look as below. Each cell contains the number of predictions made for that class, as well as the percentage that the number is of the given whole, with the aim to maximize the values on the major diagonal. Confusion Matrices are important because they allow insight into which of the types of error that are happening and with more than one, how changing parts of the system affects these values, all with one figure. Another reason why they are a good tool is that they can be extended to multi class classification (i.e. determining the kind of attack) with ease and the same specificity of being able to see which classes are being misidentified as the other. While the precision and recall statistics can display the former, the confusion matrix is far superior when the problem exceeds a binary classification, as the extra specificity it provides is invaluable.

Table 2. An example confusion matrix for binary classification

		Predicted	
		Malicious	Benign
Actual	Malicious	True Positive	False Negative
	Benign	False Positive	True Negative

Table 3. An example confusion matrix for a multiclass (specifically 5) classification

				Predicted		
		0	1	2	3	4
Actual	0	TP				
	1		TP			
	2			TP		
	3				TP	
	4					TP

2.2.4 Relevant Machine Learning Strategies

2.2.4.1 One Class Support Vector Machine

A One Class Support Vector Machine [21] (OCSVM) uses a kernel function to create a separating hyperplane around what then training data models as normal behaviour. New data is then able to be classified as normal or abnormal by whether it falls outside of this boundary or not. Common kernel functions can be classified into two groups, linear and non-linear, with the non-linear kernels used for data that has overlapping class boundaries or that is not easily separated by a linear kernel. In SciKit-Learn [22], the implementation is based on libsvm [23], even if the kernel choice is linear. OCSVMs are trained in an unsupervised manner, which does not require labelled data to train.

Table 4. Possible Kernel choices with mathematical formulation and linearity

Kernel	Mathematical Formulation	Kernel Type
Linear	$k(x_i, x_j) = x_i^T . x_j$	Linear
Polynomial	$k(x_i, x_j) = (x_i . x_j + 1)^d$	Non-Linear
Radial Basis Function (RBF)	$k(x_i, x_j) = \exp\left(-\gamma \left\|\left\|x_i - x_j\right\|\right\|^2\right)$	Non-Linear
Sigmoid	$k(x, y) = \tanh(\alpha x^T y + c)$	Non-Linear

The Linear kernel is the simplest available kernel that results in a simple linear decision boundary when the separating hyperplane is plotted graphically. The Polynomial kernel is similar to the Linear kernel, but instead will have a curve of degree d as the decision boundary, allowing for a greater resolution and better separation of data that is overlapping or not easily separated linearly. The RBF kernel results in curved, circle like 'blobs' that surround regions of points, which allows for the expression of complex decision boundaries, at a higher risk of overfitting to training data. The Sigmoid kernel is used as a rough approximation for a Multi-Layer Perceptron neural network, with two layers.

2.2.4.2 Linear Support Vector Classifier

A Linear Support Vector Classifier [24] (LSVC) (as defined by SciKit-Learn) is a special case of the generic SVC. Whereas the SVC takes the same common kernels as the OCSVM, the LSVC is restricted to linear kernels, as it is based on a different library than the SVC, being liblinear [25] rather than libsvm. This does make a difference in practical terms, as it allows extra loss functions and penalties to be explored and has better scaling with large numbers of samples than the using the SVC with a linear kernel. The LSVC is a supervised method, meaning it requires accessed to labelled data for training.

The LSVC also allows for inherent multiclass classification with the use of the "crammer_singer" multiclass strategy, based on the work of Crammer and Singer [26]. This approach removes the need for having n one-vs-rest classifiers, as it optimizes a joint objective function across all of the classes, at the cost of more expensive computation (it must solve the dual problem formulation).

2.2.4.3 Isolation Forest

An Isolation Forest [27] is an ensemble classifier, that uses many so called "Isolation Trees" as its way of classifying whether the given point is normal or abnormal. The reason for using a forest rather than a single tree is that this increases the diversity and reduces biases that may be inherent to a tree trained only on one part of the feature space. The Isolation Trees correspond to splitting up the data geometrically, until the partition has only a single value, or all the same values. Points that are different to the rest of the data are easier to separate using this method, as they lie further away from the main cluster(s) of normal points. Once the forest has been grown, new data is passed through the forest, with points that are abnormal taking much shorter paths through the trees than the normal observations. An Isolation Forest is an unsupervised learning method.

2.2.4.4 K-Nearest Neighbours

K-Nearest Neighbours [28] is a method that relies on similar points being of the same class. If the point is plotted in N-Dimensional space (where N is the number of features that describe a single point), it will likely be close to other points. Of these, K are chosen (with K being fixed) and determined what class of point they are, with the new point being classified as whatever the majority is of its K neighbours. K-Nearest Neighbours is a supervised learning method, as it is required to know what classes the points are when the system is being trained for the algorithm to function.

2.2.4.5 Local Outlier Factor

Local Outlier Factor [29] (LOF) works in a very similar manner to K-Nearest Neighbours, with new points being given a "Anomaly Score", which represents how different it is with respect to the neighbourhood of points that it lies in. This score is then compared to that of its neighbours, with anomalous points receiving lower scores, as they do not tend to cluster together as much as the normal points, with anything below a certain threshold classed as an outlier. LOF is an unsupervised training method.

2.2.4.6 AdaBoost

AdaBoost [30] is a meta-strategy, where it does not actually represent a particular learning algorithm of its own. Rather, it is a method of improving the performance of other, weaker, classifiers by training them in succession, where the output of each one is used to adjust the performance of the next. The stack of classifiers improves in performance the deeper it goes, as each successive one is trained to do better at what the previous was weak on. In this case, the default weak classifier of the decision tree will be used.

2.2.4.7 Random Forest

A Random Forest [31] is another ensemble-based classifier like the Isolation Forest, though this time the algorithm is building Decision Trees rather than Isolation Trees. To help control overfitting and improve diversity, each tree is made from only on a sample of the dataset, rather than the whole on each and every time. Once the forest of trees has been constructed, the forest then classifies new samples by running it through each tree, and whichever class

gets the vote from most trees will be the one assigned. This is superior to using a single large tree, as it allows more of the features to influence the decision more and reduces biases as each tree is constructed randomly from a sample of the data. Random Forests have inherent support for multiclass classification.

2.2.4.8 Logistic Regression

Logistic Regression [32] seeks to form a decision function based on optimizing the cross-entropy loss across the classes as it trains. This is comprised of a coefficient for each feature and each class present in the dataset, along with an independent bias term. How well new data fits the learned function determines the probability that it belongs to that corresponding class, with the best fit being the choice of the classifier. Logistic Regression has inherent multiclass classification abilities.

2.2.4.9 Gradient Boosting

Gradient Boosting [33] is much like the AdaBoost meta-strategy. It begins from a single weak learner (in this case, a Decision Tree Regressor) and then over each iteration, it trains more of these trees and continues to optimize the performance using a differentiable loss function (deviance is used, to make it a form of optimized Logistic Regression). The overall number of iterations will affect how well the model can classify the data. Gradient Boosting used in this manner is an inherently multiclass classification strategy.

2.2.5 Summary

In short, there are a number of known possible attacks that can be performed against the in-vehicle network that have been shown to be effective on the target vehicles. The difficulties in the resource constrained, real-time environment make traditional security techniques such as encryption or authentication mostly non-viable, though there has been some work towards implementing these. The majority of the existing systems instead utilize a form of intrusion detection system, which can be based on a number of different techniques, with the most common being rule-based and machine learning based approaches. The algorithms used are often neural networks, which are able to achieve high detection rates with varying amounts of false positives, though often rather low. These methods are not without their drawbacks, such as larger amounts of resources needed, being limited to specific attacks or techniques, or requiring certain characteristics such as cyclical messages to function. However, there exists the option of applying other machine learning methods, both for the detection and for the classification of the detected attacks, the latter of which no other work has done so far.

3. Research Methods

3.1 Overview

This section will detail the research methods that are relevant to this work and why that is the case for each method used, considering the relevant factors in each case. It will also cover why these methods are important in the context of this particular work, rather than in a general sense.

3.2 Qualitative Methods

The sole qualitative method employed is detailed below, as the nature of this work makes it such that most of the qualitative methods (such as conducting studies with users, focus groups and such) do not apply in this case. This is because the performance of this system is better analysed by using quantitative methods, as it is very heavily focused on numerical aspects and problems. It would be possible, after implementing a system to conduct these kinds of activities (with relevant ethical considerations and approval obtained by the appropriate parties), but this would not actually tell us about how the system was performing. It would instead be more about how users perceive and interact with the system, and in this case such interaction will be extremely limited, so it would be better to focus more heavily on quantitative methods. The most applicable method that has not been used would be phenomenological research, where a group of participants used the system as it was intended for an extended period of time and then gave their feedback, though this becomes of more value when the system is more than proof of concept.

3.2.1 Reading Existing Literature

This is an extremely important method for this particular project, as there is already a variety of existing work that covers systems for increasing awareness of the in-vehicle network. If these systems already have been proposed and could exist, it would be redundant to cover the same ground again, so it is necessary to have a thorough understanding of these systems as to avoid doing this. The same understanding is also used to be able to actually design the system subject to the specific requirements of the environment it is intended to be deployed in. This is also important to provide information about the kinds of attacks that need to be detected and classified and the methods by which others in the field are doing so (for detection at least). The more attacks that can be known about along with their corresponding indicators of compromise, the more effective and comprehensive the proposed system can be, both in the detection and classification of the attacks.

3.3 Quantitative Methods

Quantitative methods are much more applicable to the kind of work that aims to be covered. This is simply because the best way to assess the system that is created against its peers to determine whether any improvement has actually been generated is by comparing against the statistics that they have reported in their own work. The metrics detailed in the following section should provide a more than comprehensive form of coverage for both the final results obtained, as well as for tuning and optimizing the chosen implementation(s) during development.

3.3.1 Correlational

The aim of this work is to be able to establish a link between the CAN data and being able to identify the trends and patterns that correspond to the various kinds of cyber-attacks that may be performed against the in-vehicle network. As such, this piece of work is correlative, as the work is exploring the correlation between the various items of data inside of the CAN frames along with other attributes about the traffic to be able to use for our purposes. Any work that is done without having to manipulate the dataset at all would then fall under studying the variables as they would occur in the real environment (as the datasets have been collected from the operation of a real vehicle). The relationships and distribution of the variables are then studied, without actually changing the items of data to see the effect it has on the system. It is also not an aim to prove why a particular ID or data payload, or another variable causes the effect that it does, as the work is focused on being able to identify that it is X type of attack. Considering the context as well, it would be of limited use to be able to link specific CAN ID and payload combinations to always being attacks, as these mappings will most likely be specific only to the vehicle that they were collected from, as the CAN standard defines nothing about the content of the frames. Considering spoofing attacks in particular as well, the combination may be an attack in one case, but a perfectly valid frame in another.

3.3.2 Causal-Comparative

This work may also be considered as a piece of causal comparative research, as it is possible to identify specific variables and how they link to the dependent one (whether the frame is an attack or not). This is different to the methods that are correlational in nature, as it is not so much examining the system as a whole, but rather specific parts of it. In the case where specific rule-based components are used, this is very much a causal comparative approach, as the rule pertains to a single variable (though it is not directly manipulated) and the change in detection performance observed. The analysis of the system that is built when compared to the hypothesis will also be of this kind of research. It is the intent of the work to draw conclusions and offer discussion around both the data that is produced, and that of other similar papers. This will be done considering the variables identified, as well as offering discussion on the ones that may not have been that may still be relevant to the outcome of this kind of system. As such, by offering this kind of analysis, there is an attempt to be able to link the variables with their outcome, the causation, rather than just sticking to strict correlation.

3.4 Summary

This section covered the three kinds of relevant research methods, reading existing literature, correlational and causal comparative research. For the first, this was used because there is no reason to repeat content that other work already has covered, as well as using this work to guide the direction of this work to attempt to achieve the best possible results. Correlational and Causal Comparative methods are used because the easiest way to analyse and compare between this work and the existing work is through methods of this type, namely metrics such as were covered in section 2.

4. Implementation

4.1 Overview

This section covers the implementation details. This includes the three-step model of the system overall; the high-level choices made such as language and framework as well as the justifications for using the machine learning algorithms that were covered during section 2.2.4. It will also cover the datasets and the attack model that they represent, to provide the appropriate context for the system, in addition to providing explanations of how the rule-based components operate.

4.2 System Model

For the system that was chosen to be implemented, it was clear that it would need to be at least a two-stage process, consisting of identification of attacks and classification at the very minimum. However, the method that showed to have the best performance from the literature was already a combination system, of rule based and then a ML component (a DNN). As such, a three-step model was chosen which is outlined below in the diagram. Additional details for each stage are available under the appropriate subheading inside of this section.

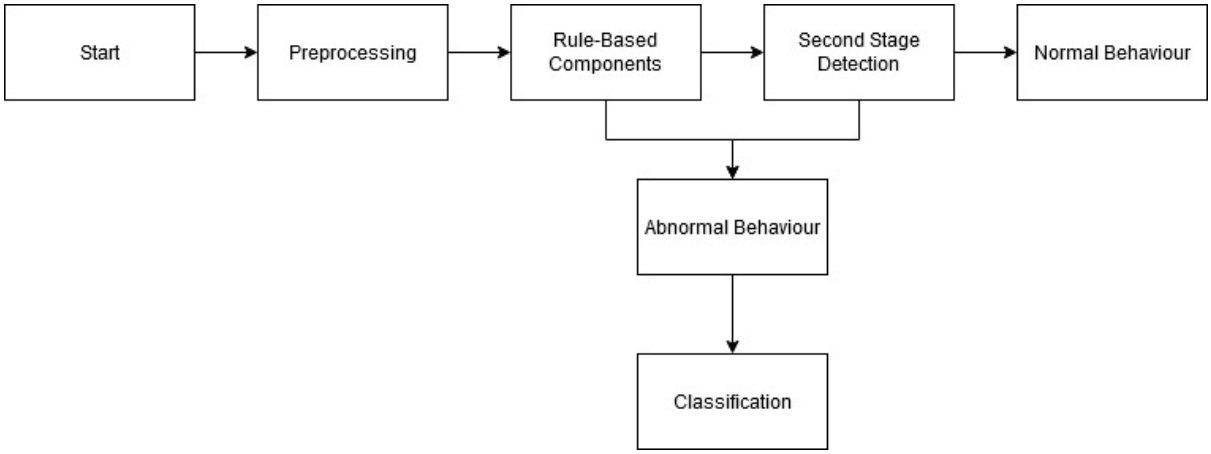


Fig 2. Outline of the proposed system with the flow between stages shown.

The proposed model differs from that of existing methods in that it is the first work to attempt to classify the attacks that it detects into the various types that are known to be present in the dataset. It also is not using a neural network-based approach, which differs from the majority of the existing work that is present, but instead seeks to apply more simple machine learning algorithms in the aim to achieve similar or better performance in the areas of detection and false positive rates, while having a lower memory usage and prediction time per message than the existing approaches and offering the ability to be altered and scaled to suit all kinds of in-vehicle networks.

4.4.1 Preprocessing

At the preprocessing stage, three main things occur. Were the system were deployed in the real environment, the messages would be assembled into the pandas [34,35] dataframe format that the system is expecting. For the development however, this is loaded from a CSV file on the disk instead, that has only a limited set of the fields of the CAN frames. Of these fields (Timestamp, ID, DLC and data bytes 0-8), the DLC and the data bytes are immediately

discarded, as the system does not use these as part of the process. This is for two reasons, which are related to one another. The first is to avoid clouding the feature space and the second is that for the vast majority of frames (at least in the datasets that were used), these showed little to no variance, which means they would contribute little or nothing to a classification problem other than add extra dimensions. The final step after the data is dropped is to add the extra attributes that are derived from the timestamp and the IDs, being the time difference between the ID and the previous occurrence of the same ID, the number of occurrences of the ID within the past 1000 messages (roughly corresponding to the maximal number of messages per $1/10^{th}$ of a second) and the relative entropy of the ID with reference to an established probability distribution (as per Müter and Asaj). This reference distribution is generated from a capture of normal behaviour on the bus and to be used to maximal effectiveness, there should be a reference for every possible valid ID defined for the bus. Mathematically, the relative entropy (also called Kullback-Liebler Divergence) is defined as below:

$$RE_{(P|Q)}(x) = p(x) \log\left(\frac{p(x)}{q(x)}\right)$$

In the above formula, P and Q are the probability distributions of IDs on the bus that were observed in this period, and the reference distribution generated from normal bus traffic, respectively. x is the particular ID that is currently being computed for, with $p(x)$ and $q(x)$ meaning the probability of x in that given distribution. As entropy is a measure of the randomness of the system, increases in the relative entropy mean that there is has been a decrease in the randomness of the system which indicates abnormal behaviour. This correlates to attackers injecting messages onto the bus, as this is not a random event and so stands out very obviously using this metric (at least for spoofing attempts). Once these values have been computed for all of the frames in the dataframe, it is passed to the first of the detection components.

4.4.2 Rule-Based Components

The system uses two rule-based components, though there are the potential to use more as long as they have a low execution time, as to not compromise the real-time performance of the system. The two rules chosen are a valid ID rule and a time interval rule. The first of these is defined below.

```
Algorithm 1: ID Based Detection
Require: Preprocessed_Dataframe
Load Reference List
IDs_in_frame ← Get all unique IDs in Preprocessed_Dataframe
new_IDs ← new List
For ID in IDs_in_frame do
     If ID not in Reference_List then
          new_IDs.append(ID)
     end if
```

```
end for
detected_attacks ← new List
for frame in Preprocessed_Dataframe do
     if frame.id in new_IDs then
          detected_attacks.append(frame)
     end if
end for
return detected_attacks
```

After this ID detection is applied, the frames that have been flagged are removed from the dataframe and the leftovers passed to the second rule. This requires the use of a capture of normal traffic data to compute the average time it takes for a message for the same ID to occur twice (as a large amount of CAN messages are sent with regular intervals, their average times should be very close to this time, plus or minus a very small deviance) and store this as a file. Then, when detection is being performed, compute the time delta between the messages of the same ID and if this delta is less than the average multiplied by a certain modifier, mark it as an attack. While this modifier is adjustable, it was set at 0.5, so any messages injected at double speed or higher should be detected. This was chosen because this was the minimal injection rate reported by Young et al. to have a successful attack, contrary to Miller and Valasek, who reported messages would need to be injected 20-100 times faster. This is detailed by the algorithm below.

```
Algorithm 2: Time Interval Based Detection
Require: Post_ID_Rule_Dataframe
Require: Timing_Lookup_Dataframe
Require: Injection_Rate_Modifier
Load Timing_Lookup_Dataframe
For ID in Timing_Lookup_Dataframe do
     Post_ID_Rule_Dataframe[average_time] ←
     Timing_Lookup_Dataframe[ID]
End for
For frame in Post_ID_Rule_Dataframe do
     If frame.time_difference > 0 and frame.time_difference <
     frame.average_time * Injection_Rate_Modifier then
          frame.is_attack ← true
     end if
end for
```

```
return Post_ID_Rule_Dataframe[is_attack] = true
```

Once this is complete, the detected attacks are then removed once again, the leftover frames being passed to the second stage of attack detection, which is machine learning based.

4.4.3 Second Stage Detection

This stage of the process uses machine learning algorithms to try and identify attacks that the rule-based approach would not be able to catch. It is important to go beyond a simple rule-based system as while these are lightweight and much faster than more complex approaches, they also have the inherent limitation that their detection ability is only as good as the rules that are defined. As such, it is important to be able to identify attacks that fall outside of the rule profile, as there are no guarantees that attackers will use an attack that known about and has a well-defined signature. The aim of the systems at this stage is to be able to identify whether a frame is malicious or not, regardless of what kind of attack this frame corresponds to, making this either a novelty detection or binary classification problem.

4.4.3.1 One Class Support Vector Machine

The OCSVM was chosen as a method to test as it is an unsupervised detection method, meaning it can be deployed without having access to any instances of attacks. Considering that any system being deployed in a real context would need to be adjusted for the specific IDs in use on the vehicle, this gives it an advantage as it can be trained by just capturing a sample of normal bus traffic (for which there exist commercial products to do so). OCSVMs are also a relatively lightweight system when compared to neural networks and are easier to train. Dependent on kernel choice, the OCSVM should also be fast enough to perform in a real time context with the appropriate number of samples for a single vehicle.

4.4.3.2 Linear Support Vector Classifier

The Linear SVC was chosen for two reasons. One, if a lightweight linear model can achieve comparable performance, there would be no need to use more expensive or slower methods. Two, it scales much better to larger amounts of training data than a comparable linear SVM based on libsvm, so can be tuned, and trained more effectively as more combinations of parameters can be tried for the same time frame.

4.4.3.3 Isolation Forest

The Isolation Forest method was chosen because it should be able to achieve a very high performance while managing to still be very fast as working with new data. Using a relatively large ensemble of trees also helps to average out the variance between them, much like in the random forest. However, bias in the Isolation Forest is bad, though not able to be avoided, due to the algorithm (though there is “Extended Isolation Forest” that can help alleviate this problem, it is not available inside of Scikit-Learn). The implementation inside of Scikit-Learn also has an additional parameter, contamination, that expresses the amount of expected abnormal points inside of the data. This allows fine control over the classifier performance, though fixing it may prove problematic.

4.4.3.4 K-Nearest Neighbours

K-Nearest Neighbours was included as part of the experiment for several reasons. The first is that (depending on the value of k), it is a relatively fast algorithm to compute, which would be needed for a real time context. It also does not really require much tuning or adjustment,

as long as the data going in is representative of what the classifier will see in the future, it can achieve very good performance. Finally, in the right set of circumstances, it can also be a relatively lightweight system, though this is dependent on the number of features and neighbours used.

4.4.3.5 Local Outlier Factor

LOF was included as part of the algorithms for testing, as it was another unsupervised training method and similar in nature to the K-Nearest Neighbours algorithm, at least theoretically. Therefore, it was included to determine whether it could be used in a similar manner, without first requiring labelled training data, as this needs to be produced vehicle specifically.

4.4.3.6 AdaBoost

The AdaBoost algorithm was included because far as can be told, no one has yet used this particular technique for this specific application. It is also a meta-strategy, which none of the other techniques are, so it will provide a useful reference point when comparing to the other supervised learning methods.

4.4.4 Classification

At this stage, all of the frames that were detected by the rule-based and second stage are passed to a classification algorithm to be able to determine what class of attack was taking place. This can be significant in that it would allow a system to go beyond just knowing attacks are taking place, to (with future work) being able to use active countermeasures against the attacker to try and stop whatever attacks are taking place. It also allows knowledge of which areas are being targeted and need to receive additional security updates or measures to harden them against the actions of attackers such that future attacks of the same kind are ineffective or more easily mitigated. Classification of attacks in this manner is a multiclass classification problem and as such, it is better to use algorithms that inherently support multiclass classification, rather than ones that need to use one vs rest or one-vs-one approaches, as there are potential scaling issues in terms of memory and compute time when there are large numbers of classes.

4.4.4.1 Linear Support Vector Classification

The Linear SVC is a desirable choice of algorithm as it both very lightweight and fast to make predictions. Through the use of the crammer_singer multiclass strategy, it can also be used as an inherently multiclass classifier, which avoid the need to train large numbers of classifiers to model binary problems when there are larger numbers of classes involved in the system.

4.4.4.2 Random Forest

The Random Forest is a desirable choice of algorithm here as it is extensible (should more features than the ones used here wanted to be added), and yet even with a high dimension problem, the classifier should not see a degradation in performance. As each of the trees is constructed from a small subset of both the feature space and the dataset, there is a low amount of bias present in each tree. This comes as the cost of a higher variance, some (though not all) of which is averaged away by using a forest of trees, rather than a single one. Once trained, the classifier is also extremely quick to work with new examples, which is a critical factor when it comes to a real time system.

4.4.4.3 K-Nearest Neighbours

K-Nearest Neighbours functions no differently when used as the classifier here as it would for just detecting attacks at the second stage. As such, details will not be repeated here, though it is suited well for classifying the different classes of attacks because a lot of the similar, normal points will cluster together and then the attack classes will either form their own clusters (for spoofing and DoS) or be much more spread out (in the case of Fuzzy attacks) in the chosen feature space.

4.4.4.4 Logistic Regression

Logistic Regression was chosen to be included because it is a very simple technique to apply than can achieve good results on some data. If it proves to be effective, there would be no need to apply more complex techniques instead that consume more resources. As the classifier is a simple learned function, it is also extremely lightweight and very fast, with the capability to be extended well to large numbers of classes without severe scaling issues.

4.4.4.5 Gradient Boosting

Gradient Boosting was included to both have a present meta-strategy, to see if it performed noticeably better than other techniques and because of its ability to scale well to large numbers of classes and training data, while remaining performant at prediction time.

4.3 Implementation Choices

4.3.1 Choice of Language

Python was the chosen language for this for a number of reasons. The first is that the author has the most amount of experience using Python. Next, there are a large choice of frameworks that have a Python API, with more than one being able to be used at the same time thanks to being able to import specific parts of a chosen package with the Python import system. This avoids the issue of needing a specific feature that a framework did not support and having to code it from scratch, as that would be a waste of the rather limited time that is available. The only disadvantage of using Python is that the language is interpreted and not compiled, which means that is much slower than using Java or C++. However, this partially mitigated when it comes to using the available machine learning frameworks, as these are actually often Python wrappers over underlying C/C++ code that is much, much faster than pure Python.

4.3.2 Choice of Framework

SciKit-Learn was chosen for this work for several reasons. As Python was the language that was desired, this ruled out using Weka, which while it has a GUI and is (allegedly) very easy to be able to pick up and use, only has an API for Java. While this is the other language that was considered, the corresponding Java code is a whole lot more verbose and requires more effort to accomplish the same level of functionality. Weka also offers a whole range of possible options, but short of applying them all, it would be difficult to determine which of them were actually useful. SciKit-Learn provides a large choice of algorithms, while also giving access to a variety of preprocessing methods and reporting metrics, so almost all the functionality that was required could come from a single library. This both helps to keep the code lower weight as well as making development simpler.

4.4 Datasets and Attack Model

There are two datasets that will be used during this work. The first of these is the “Car Hacking Dataset” [36] and the second the “Survival Analysis Dataset for Automobile IDS”

[37], both of which are from the HCRL. The Car Hacking Dataset merely specifies “collected from a real vehicle by logging CAN traffic via the OBD-II port” but the Survival Analysis dataset gives details about which vehicles were used to collect the data, being a Hyundai YF Sonata, KIA Soul and Chevrolet Spark. This confirms that all data is collected from real in-vehicle networks and as such, should be representative of a realistic environment. Of the two, the Car Hacking Dataset is the one with more data and attack types, so this is the one that is used as part of the development of the system. From this dataset, the attack model is defined as below:

- DoS Attack – Messages with the ID 0x0000 are injected with an injection rate of 1 message per 0.3 milliseconds.
- Fuzzy Attack – Injection of messages with random ID and Data payloads, every 0.5 milliseconds
- RPM Spoofing – An injection attack targeted specifically at the RPM gauge with the aim of spoofing a false value, injections are every 1 millisecond
- Gear Spoofing – An injection attack targeting the gear information, with the aim of spoofing a false value, injections are every 1 millisecond.

From the Survival Analysis Dataset, define the following attack types:

- Flooding Attack – Inject a large quantity of messages of ID 0x0000. Near identical to the DoS attack from the Car Hacking Dataset.
- Fuzzy Attack – Identical to the one in the Car Hacking dataset, but with an injection rate of every 0.3 milliseconds.
- Malfunction Attack – Target a specific CAN ID and simultaneously manipulate the data payload and inject messages of random CAN IDs.

This leaves five overall types of attack across both datasets. Of these, the four in the Car Hacking dataset will be used during the development and training of the system, with the Malfunction Attack deliberately left out and only seen during testing to see how any implementation responds to attacks of an unknown nature.

4.5 Summary

In this section, the proposed system model was put forward, the preprocessing and rule-based components explained through what each feature would be and through use of the algorithms behind them, respectively. Also covered were the choices for using each of the algorithms laid out in section 2 and why they might be useful to test as part of this work. Furthermore, the major choices in implementation of language and machine learning framework were stated and justified with comparisons to the other options given. The datasets used were also explained, defining the DoS, Gear, RPM, Fuzzy and Malfunction type attacks with an explanation of each provided, while also explain why the Malfunction type attack will not be used as part of the training data.

5. Results

5.1 Overview

This section contains all of the results obtained from the practical components of this work. That consists of the optimization carried out to try and determine the best parameters for each of the algorithms used at both the detection and classification steps, the Precision, Recall and F1 scores of each detection algorithm, the confusion matrices for the classification algorithms and the overall memory usage and speed of each algorithm or combination of algorithms. Brief explanations or discussion will be offered for each of these, but the majority of the discussion, possible reasons for the observed results and comparison to other existing work is instead in the next chapter.

5.2 Algorithm Optimization

For each algorithm that was implemented, possible parameter values were tested to find the variants that gave the best performance. This was done using the either the ParameterGrid and a for loop, which created, trained a classifier, and then evaluated against a test set, or using GridSearchCV, which does a similar task, but uses k-fold cross validation instead of a test set (in this case, k was left at the default value of 5).

5.2.1 Second Stage Algorithms

5.2.1.1 OCSVM

For the OCSVM, there are three main parameters that affected the performance of the algorithm.

- Kernel – Determines which mathematical function the SVM uses
- Nu – What fraction of the training data can be classified incorrectly, as well as the minimum number of support vectors for determining separating hyperplane
- Gamma – Coefficient term

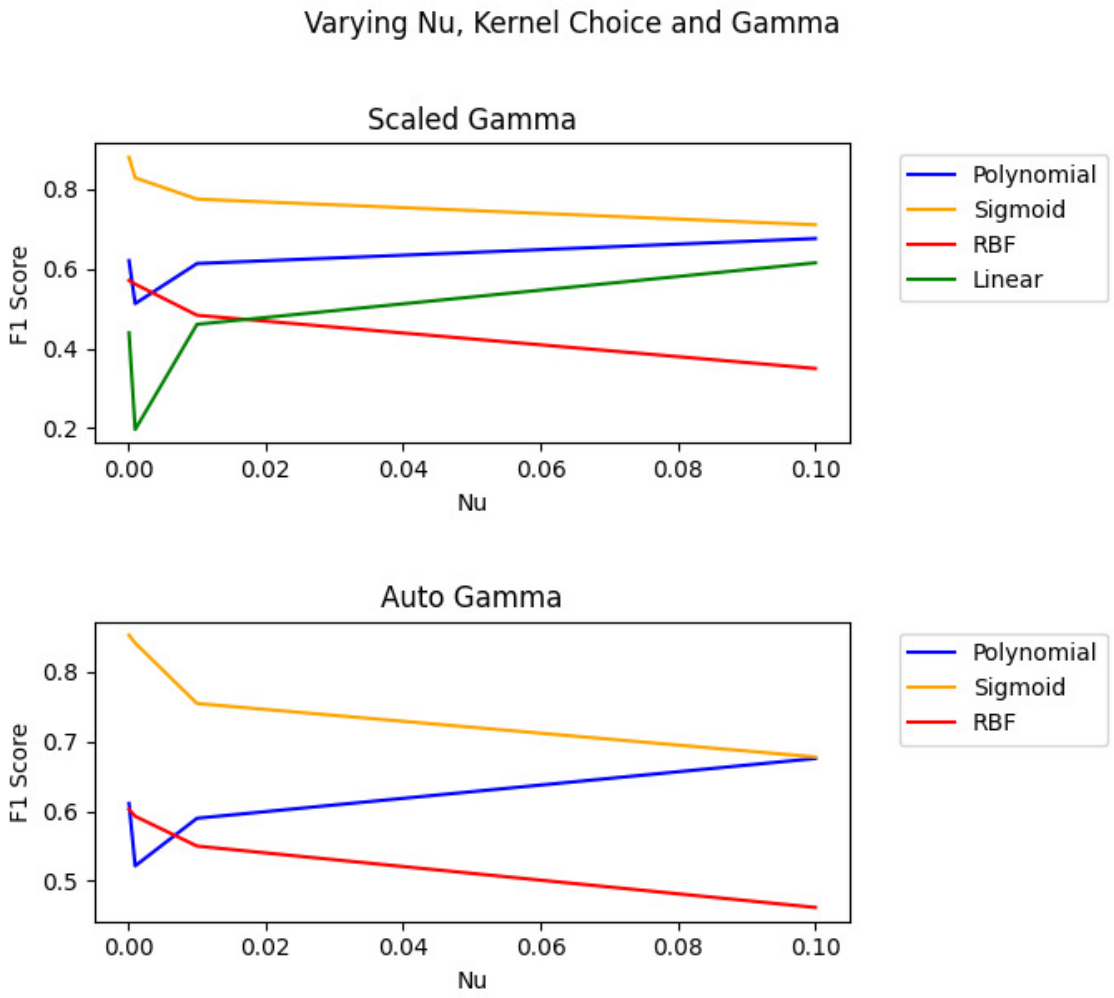


Fig 3. Graph showing the effects of varying the Nu and Gamma parameters across OCSVM kernels

The graph shows the results of the tuning of each of these parameters. When Gamma is scaled, the sigmoid kernel proves the best, with its performance in an inverse linear relationship with Nu. The RBF kernel has the same relationship, but at consistently lower F1 scores. The Polynomial and Linear kernels show an initial sharp drop at nu=0.001, but then slowly increase in performance as Nu increases again, though do not reach the same F1 score as the sigmoid kernel.

When Gamma is determined automatically, the sigmoid kernel has a similar trend to when Gamma is scaled, while remaining the kernel that achieves the best F1 score. The RBF kernel performs almost identically, with the same consistent downward trend as Nu increases. The Polynomial kernel also has the same initial drop, then slow increase in performance as Nu increases again.

From this, the chosen parameters were:

- Kernel – Sigmoid
- Nu – 0.0001
- Gamma - Auto

5.2.1.2 Linear SVC

For the Linear SVC, there are two parameters that were changed from their default values.

- Dual – Determines whether the dual or primal optimization problem is used when the algorithm is solved. As the dual problem requires finding a number of parameters equal to the amount of training data and the primal problem only a number equal to the dimensionality of the input vectors, the dual problem massively increases runtime for the large amount of training samples used and should be avoided.
- Class Weight – This applies a coefficient to the class, biasing the position of the learned hyperplane to better classify classes with higher bias.

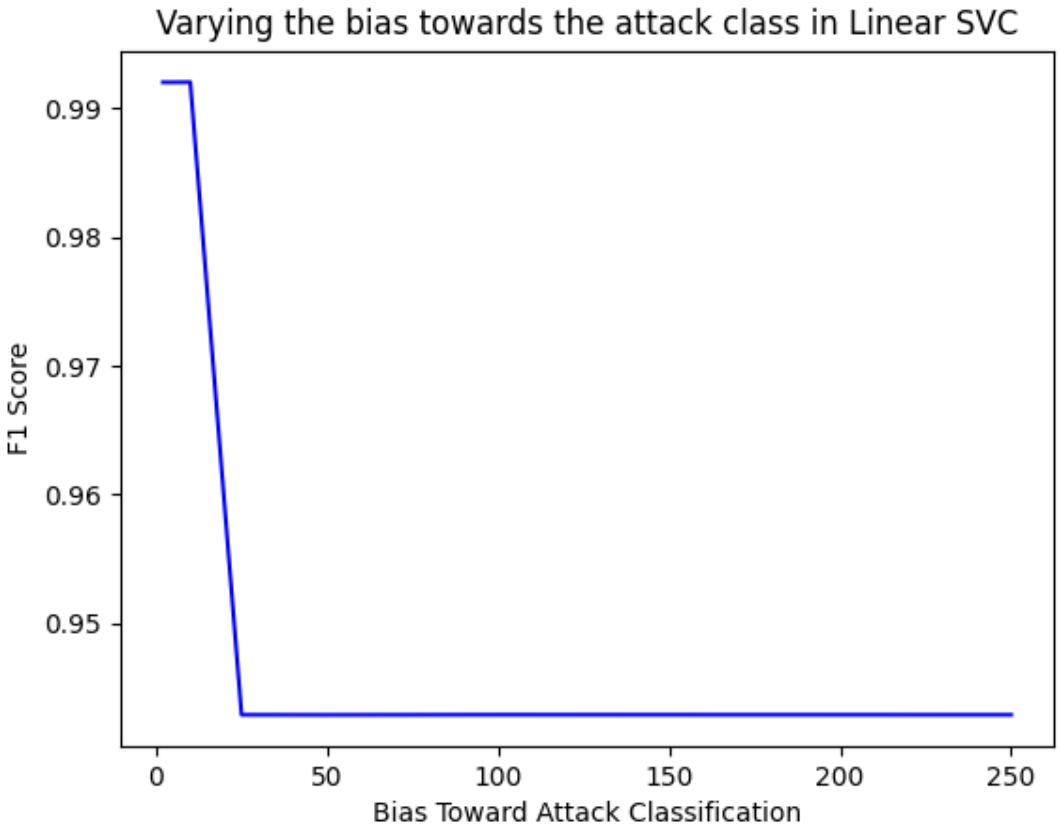


Fig 4. Graph showing the effect of varying the bias towards the attack class on the F1 score achieved.

The values determined here are able to achieve a performance that suggests the data is linearly separable to some extent, as it achieves a performance exceeding that of a naïve model on the development data. However, it fails to achieve a score near to that of the other classification methods, suggesting that to properly determine the class of a new message, the decision is more complex than a linear boundary can define.

From this, chosen parameters were:

- Dual – False
- Class weight – {0: 1, 1: 7} (seven times bias toward attacks)

5.2.1.3 LOF

Local Outlier Factor required the setting of three parameters, though only one of these was varied experimentally, n_neighbors.

- N_neighbors – The number of points to examine when determining the outlier score of the point
- Contamination – This is a measure of how many of the training points are outliers themselves. As LOF is an unsupervised method, it was trained using only normal data, but the value for contamination must be strictly greater than zero. As a result, a tiny fraction of the training data would have been considered an outlier, as the contamination was set at 0.00000000001.
- Novelty – This determines whether the LOF instance is being used for novelty detection, or outlier detection. As the algorithm is trained and then applied to new data, this must be set to true.

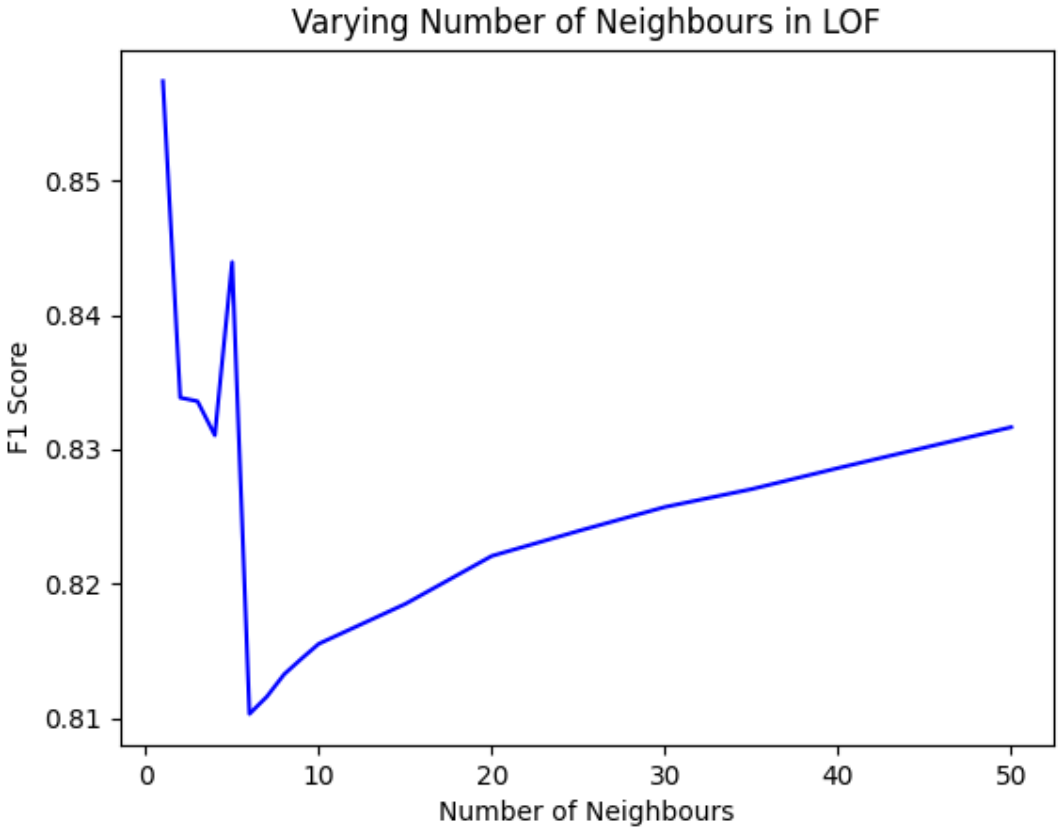


Fig 5. Graph showing the effect of varying the number of neighbours used in LOF on the F1 score.

As can be seen in the graph above, the number of neighbours used actually should be set low, as this gives the best performance when very few points are considered. The performance does start to rise again, but overall the algorithm performed very poorly when compared to any of the supervised techniques.

From this, chosen parameters were:

- N_neighbours – 4
- Novelty – True
- Contamination – 0.00000000001

5.2.1.4 IF

When tuning the Isolation Forest, only a single parameter was varied:

- n_estimators – how many Isolation Trees are created in the forest

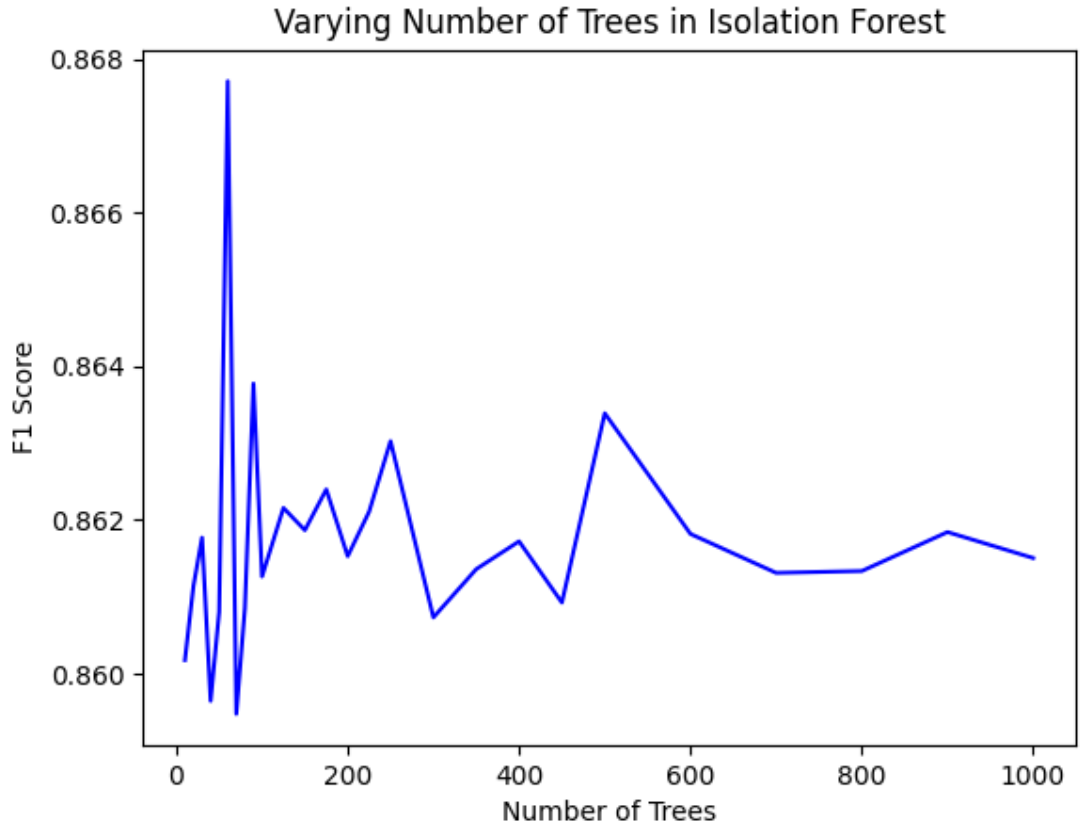


Fig 6. Graph showing the effect of varying the number of trees in an Isolation Forest on the F1 score.

The above graph shows the results of varying the number of trees that make up the Isolation Forest. From it, there does not seem to be any correlation between the size of the forest and the F1 score, which would suggest that the differences are stemming from the random samples drawn from the dataset to construct the forest, and not from a lack of trees in it.

From this, parameters chosen were:

- n_estimators – 70

5.2.1.5 K-Nearest Neighbours

For the K-Nearest Neighbours algorithm, the two parameters that were varied were:

- N_neighbors – This determines how many points are examined to classify each new point.

- Weights – The two options are to weight every point equally, or to allow the weight of each point to vary as an inverse of the distance to the point needing classification (close points are weighted strongly, far ones less).

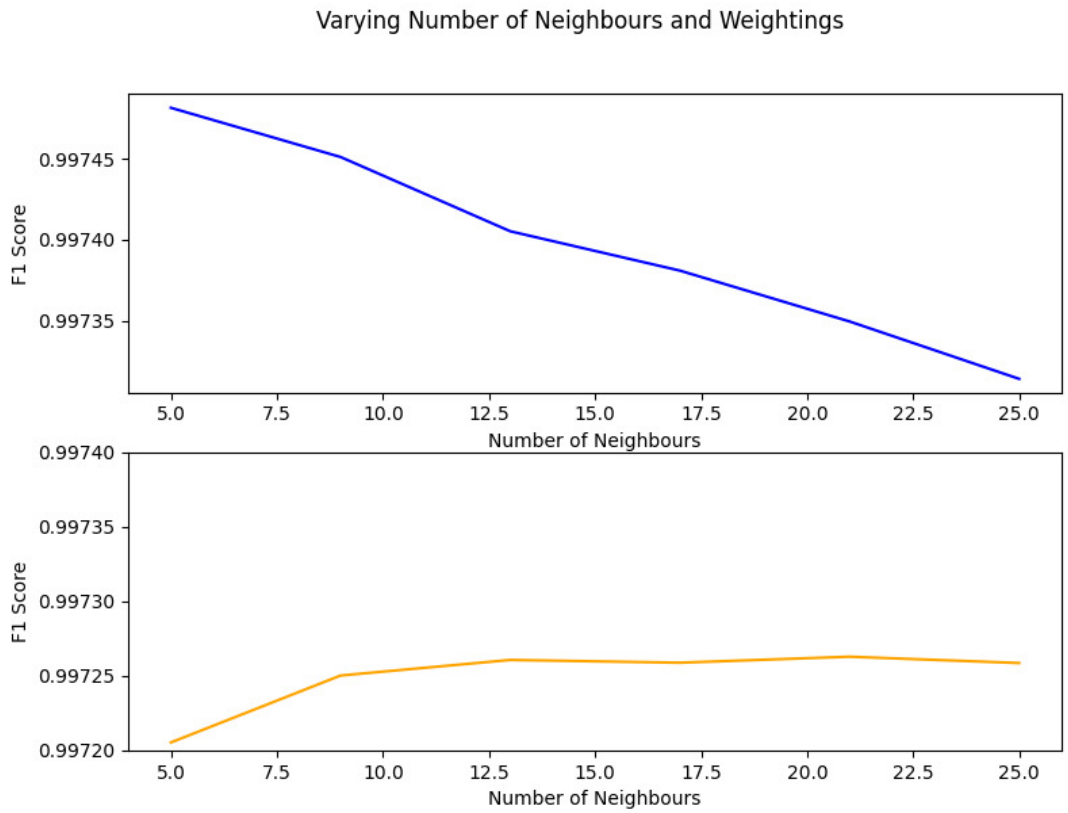


Fig 7. Graphs showing the effect of varying the number of neighbours and weighting strategy on the F1 score.

The graph above shows the effect of varying these two parameters, with the uniform weighting in blue and the distance-based weighting in orange. As can be seen, the uniform weighting system has superior performance to the distance-based equivalent for the same number of neighbours, though the difference is not that large. For larger amounts of neighbours, the uniform weighting option decreased in F1 score (which is expected), as this would allow points that are not as close in the neighbourhood to influence the outcomes just as much as points that are extremely close, so the whole system becomes less specific. The distance-based weight option sees the opposite, as the effect from the uniform system does not apply, instead allowing more closer points to correctly influence decisions and further points to have a much lesser effect.

From this, chosen parameters were:

- N_neighbours – 5
- Weights - uniform

5.2.1.6 AdaBoost

For the AdaBoost meta strategy, there is a single parameter that was used to tune the performance:

- N_estimators – how many rounds of weak classifier that the algorithm trains

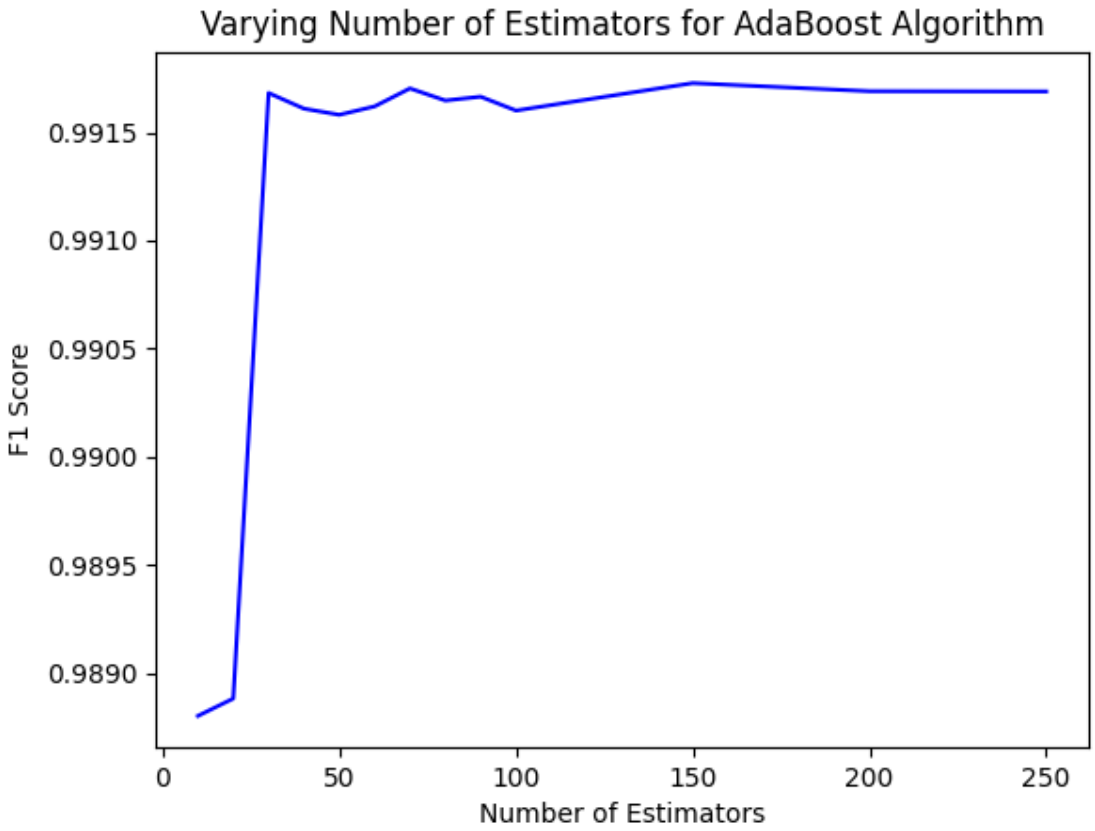


Fig 8. Graph showing the effect of varying the number of estimators used in the AdaBoost algorithm on the F1 score.

As can be seen in the above graph, the performance takes a sharp spike at around 30 estimators, then as the number of estimators increases from there on out, there is very little improvement to the performance, even when there are more than 5 times as many estimators.

From this, chosen parameters were:

- N_estimators – 30

5.2.2 Classifier Algorithms

5.2.1.1 Linear SVC

The changes made to the parameters for the Linear SVC was that of using the "crammer_singer" multiclass strategy, to change the system to an inherently multiclass classification method, as well as using the maximum value of max_iter that liblinear supports. (even though the solution was not converged after this number was reached).

Chosen parameters:

- Multi_class – crammer_singer
- Max_iter – 100000

5.2.2.2 Gradient Boosting

For Gradient Boosting, a single parameter was varied:

- n_estimators – Number of gradient boosting iterations to conduct

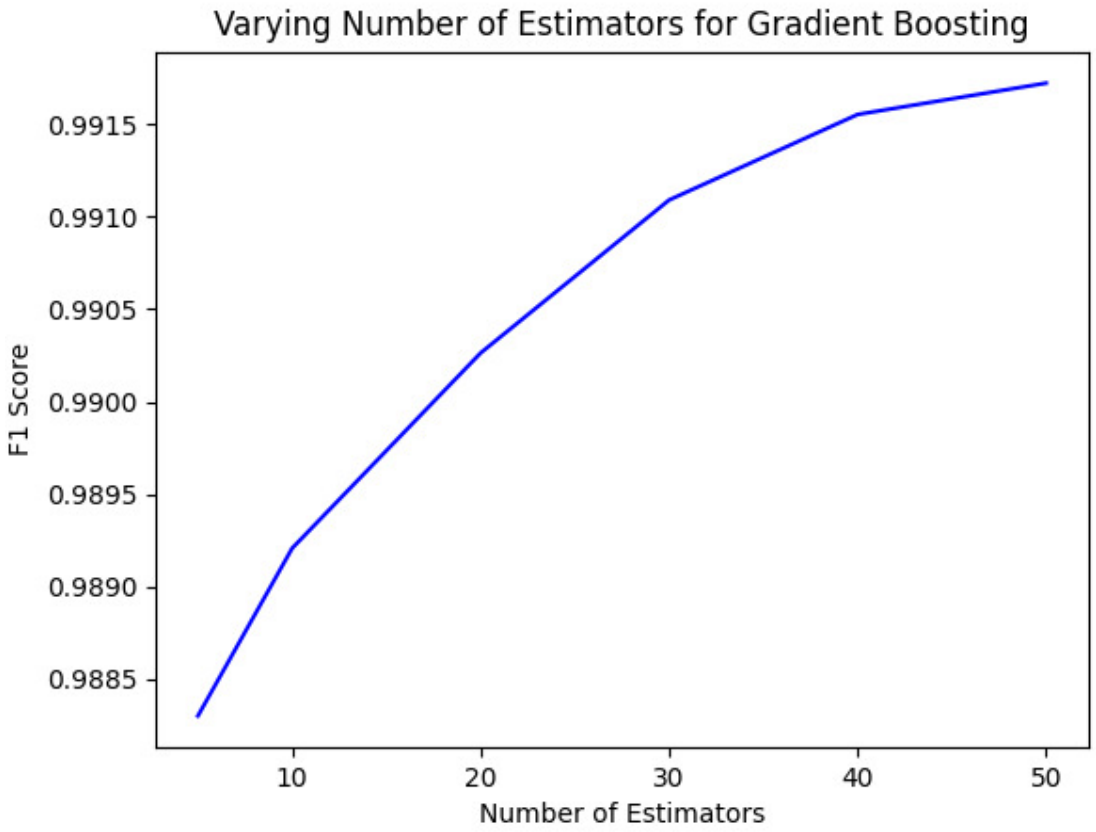


Fig 9. Graph showing the effect of number of Gradient Boosting Iterations on the F1 score for classification.

The graph above shows the effect of adding more iterations to the gradient boosting algorithm. There are consistent performance increases as the number of iterations increases, though the gains do start to become less as the number of iterations increases.

From this, chosen parameters were:

- n_estimators – 50

5.2.2.3 K-Nearest Neighbours

The K-Nearest Neighbours algorithm at this stage is identical to that for the one used in the previous step, so shares the same tuneable parameters, even though the algorithm is now being used for attack classification. As such, refer to section 5.2.1.5 if needed.

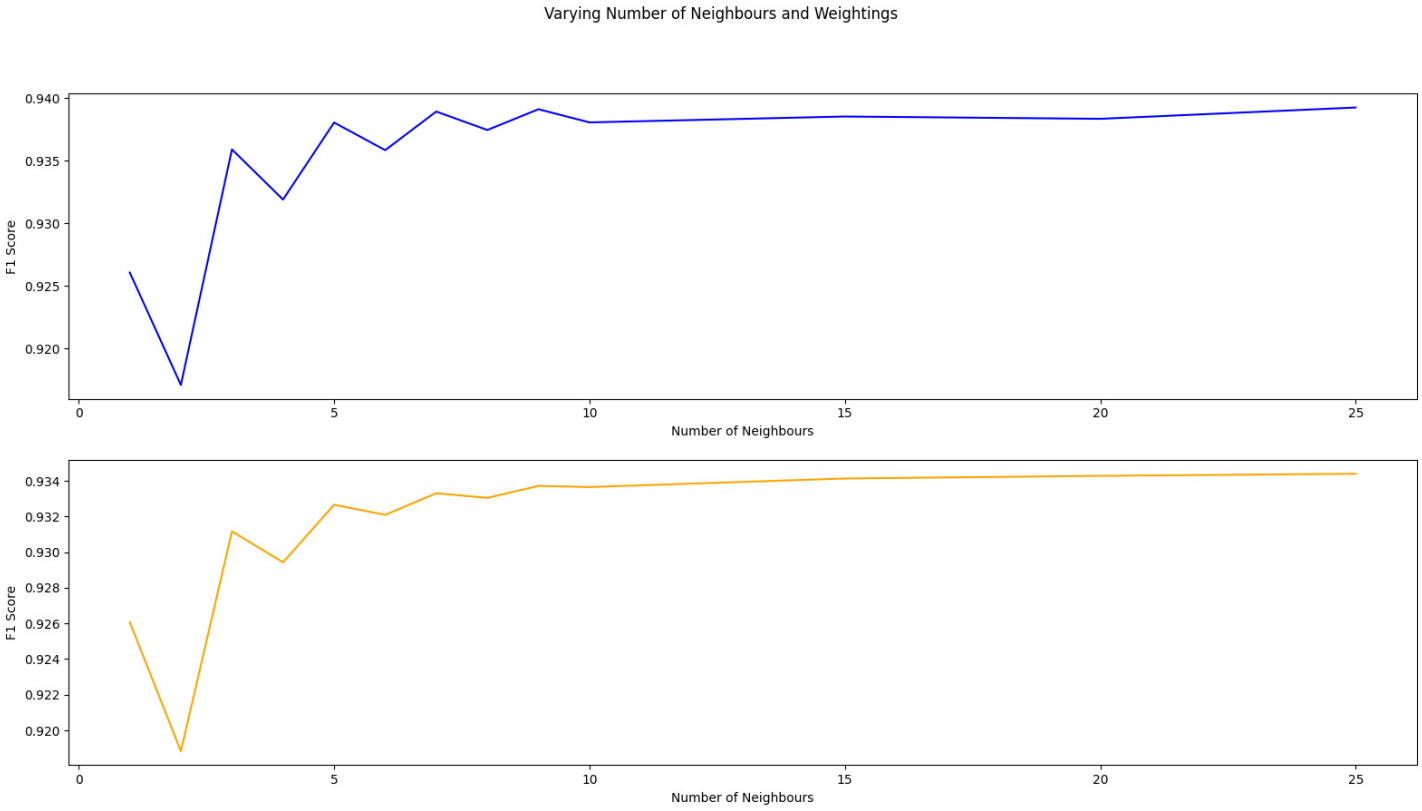


Fig 10. Graph showing the effect of varying number of neighbours and weighting strategy on the F1 score.

The above graphs show the effect of varying the same two parameters as were varied in section 5.2.1.5. The blue graph is for the uniform distance option, and the orange is for the weighted option. Examining the two graphs, they both show almost exactly the same trend, only minor variations between the two, save for the fact that the uniform distance option (nearly) plateaus at a higher overall F1 score.

From this, chosen parameters were:

- n_neighbors – 10
- weights – uniform

5.2.2.4 Random Forest

For the random forest algorithm, a single parameter was tuned:

- n_estimators – how many trees make up the forest

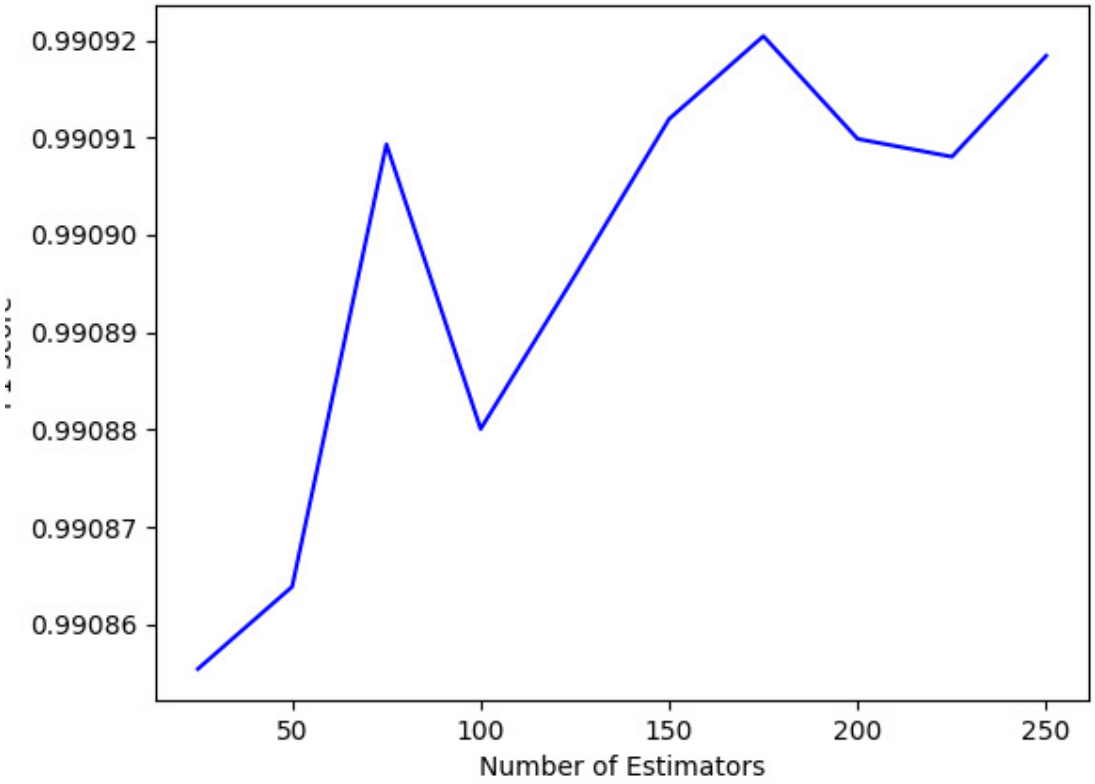


Fig 11. Graph showing the effect of the number of trees in the Random Forest on the F1 score.

As can be seen in the above graph, even with a small number of trees (the line starts at only 10), the algorithm achieves a very high performance on the optimization data. As the number of trees in the forest increases, the performance improves, though only very slightly, with no clear correlation between increasing the number of trees and the performance improvement. This variation may be due to the randomly sampled points from the data used to construct each of the trees inside the Random Forest.

From this, chosen parameters were:

- n_estimators – 250

5.2.2.5 Logistic Regression

For the Logistic Regression algorithm, four parameters were specified:

- penalty – l2 or l1 type regularization
- solver – which solving algorithm to use
- multi_class – determines the multiclass strategy to use. As multiclass regression is needed, this will be 'multinomial'
- max_iter – how many iterations to run before stopping

From a few experiments, it was determined that the following parameters were the ones that have the best performance on test data:

- penalty – l2
- solver – saga
- multi_class – multinomial
- max_iter - 100000

5.3 Identification Performance

5.3.1 Stage Two Components

Below is a table containing the Precision, Recall and F1 score for each individual dataset, rounded off to the nearest 3 decimal places. These scores are not for using the algorithm alone, they are the result of applying the algorithm after the rule-based components in each case. For the Sonata, Soul and Spark sets, the attacks have been combined into a single set, rather than being tested individually.

Table 5. Precision, Recall and F1 score of all stage 2 algorithms across attack type, then combined Sonata, Soul and Spark sets. The cells in bold indicate that the performance was the best achieved in terms of F1 score for that dataset.

Algorithm	DoS	Gear	RPM	Fuzzy	Sonata	Soul	Spark
One Class Support Vector Machine	P:0.162 R:1 F1:0.279	P:0.134 R:0.987 F1:0.236	P:0.140 R:0.982 F1:0.246	P:0.129 R:0.995 F1:0.228	P:0.177 R:1 F1:0.301	P:0.156 R:1 F1:0.270	P:0.337 R:0.848 F1:0.482
Linear Support Vector Classifier	P:0.988 R:1 F1:0.994	P:0.742 R:1 F1:0.852	P:0.751 R:1 F1:0.858	P:0.989 R:0.988 F1:0.988	P:0.460 R:0.998 F1:0.630	**P:0.242** **R:0.998** **F1:0.389**	P:0.997 R:0.773 F1:0.871
Isolation Forest	P:0.233 R:1 F1:0.377	P:0.197 R:0.977 F1:0.328	P:0.208 R:0.984 F1:0.344	P:0.230 R:0.986 F1:0.373	P:0.271 R:0.998 F1:0.426	P:0.174 R:0.997 F1:0.297	P:0.280 R:0.989 F1:0.437
K-Nearest Neighbours	P:0.988 R:1 F1:0.994	P:0.878 R:0.974 F1:0.924	P:0.871 R:0.983 F1:0.924	P:0.986 R:0.991 F1:0.989	P:0.516 R:0.997 F1:0.680	P:0.158 R:0.999 F1:0.273	P:0.997 R:0.773 F1:0.871
Local Outlier Factor	P:0.210 R:1 F1:0.347	P:0.167 R:1 F1:0.286	P:0.175 R:1 F1:0.298	P:0.146 R:1 F1:0.255	**P:0.941** **R:0.995** **F1:0.967**	P:0.141 R:1 F1:0.247	P:0.229 R:1 F1:0.372
AdaBoost	**P:0.993** **R:1** **F1:0.997**	**P:0.895** **R:0.973** **F1:0.932**	**P:0.893** **R:0.983** **F1:0.936**	**P:0.988** **R:0.991** **F1:0.989**	P:0.937 R:0.997 F1:0.966	**P:0.242** **R:0.998** **F1:0.389**	**P:0.996** **R:0.775** **F1:0.872**

From the performances in the above table, some conclusions can be drawn. All of the supervised algorithms have superior performance to that of the unsupervised algorithms. As they both convert the data to an identical form and have the same preceding steps, the difference can only be from the algorithm itself.

With regard to detection rates (determined by the Recall statistic), all algorithms perform quite well. Of the supervised algorithms, the Linear Support Vector Classifier shows comparable performance on the DoS and Fuzzy datasets to the other two classifiers but has remarkably lower precision for the RPM and Gear spoofing. Of the K-Nearest Neighbours and AdaBoost methods, they show very similar results, though the AdaBoost is marginally better at every class of attack, though the difference is very small. The unsupervised

algorithms all perform very similarly, achieving Recall values often above 98% or above, at the cost their Precision. All classifiers are able to achieve perfect recall on the DoS dataset, as all the attack instances are actually removed by the rule-based component and not the second stage algorithm. This is possible because the ID of 0x0000 is not in use normally in this dataset, so is filtered out by the new ID rule.

In terms of Precision, the supervised algorithms all see very high values on the Fuzzy and DoS type attacks, with a value approximately equal to 98-99% averaged over the two attack types. On the detection of the two spoofing type attacks, the Linear Support Vector Classifier sees a drop in Precision of around 25%, K-Nearest Neighbours 12% and AdaBoost 11%. Unsupervised algorithms have a much lower Precision, meaning that they are generating far more false positives and false negatives than the supervised algorithms, by around 4 times.

5.3.2 Classifiers

This section contains the confusion matrices for when each classifier is run against the results of the first two stages (using the AdaBoost algorithm as the second detection component) for a combination of all individual attacks per dataset. As such, there will be 4 matrices, one for the Car Hacking Dataset and then one each per Sonata, Soul and Spark.

5.3.2.1 LSVC

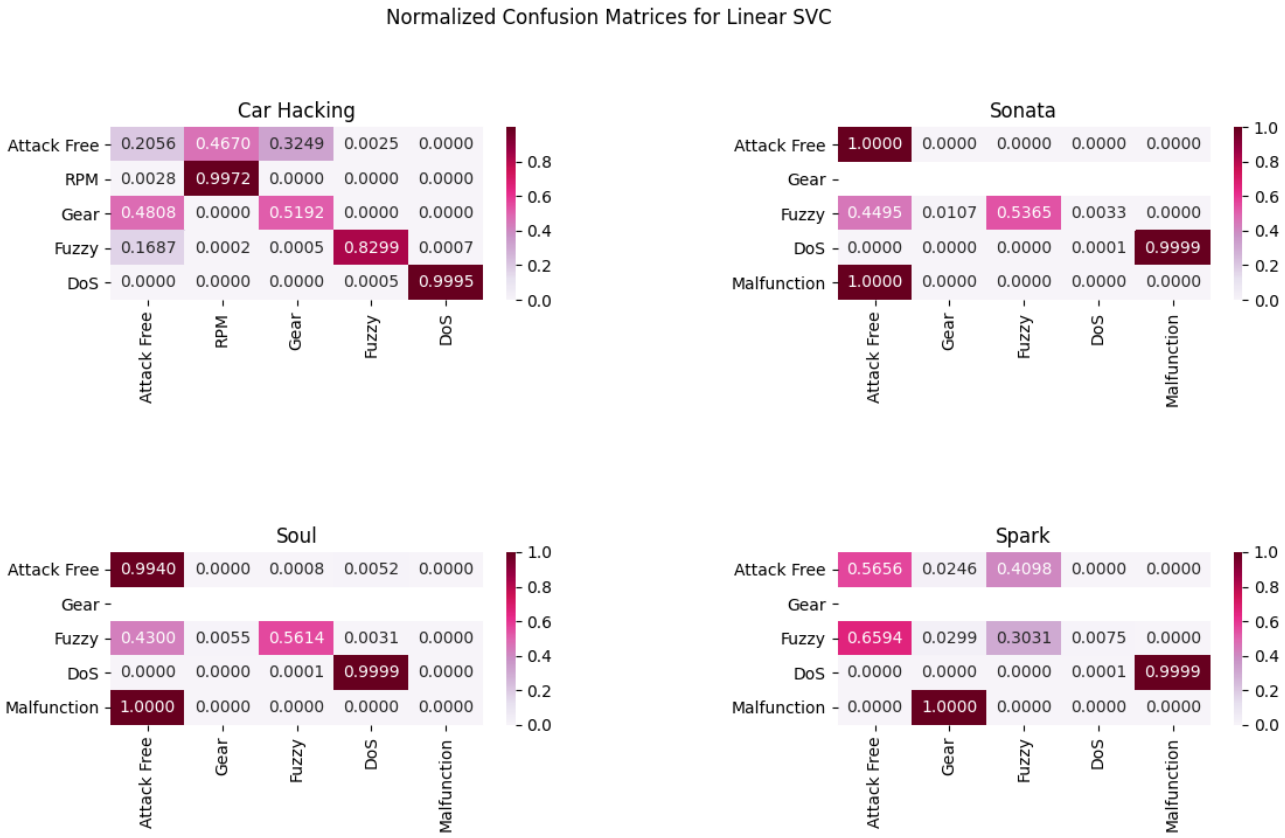


Fig 12. Normalized Confusion Matrices per dataset for the Linear SVC classification method.

In the above Confusion Matrices, the Linear SVC has very good performance on the RPM and DoS attacks in Car Hacking, good for Fuzzy in Car Hacking and is relatively poor at

classifying the remaining two classes. Performance on the Sonata, Soul and Spark sets is good for benign data, poor for Fuzzy attacks, mostly wrong for DoS attacks and completely wrong for Malfunction attacks.

5.3.2.2 Random Forest

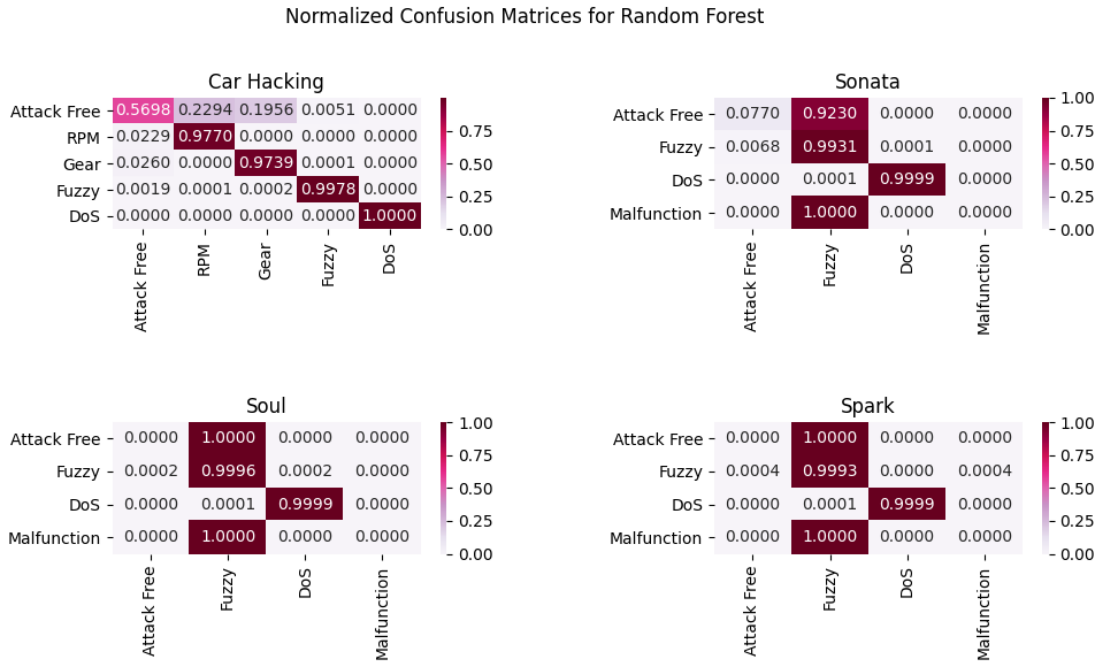


Fig 13. Normalized Confusion Matrices per dataset for the Random Forest classification method.

In the above Confusion Matrices, the Random Forest has very good performance on all attack types for the Car Hacking set, only struggling with the benign data. Performance on the Sonata, Soul and Spark sets is very good for DoS and Fuzzy attacks, while being completely incorrect for Attack Free and Malfunction attacks.

5.3.2.3 K-Nearest Neighbours

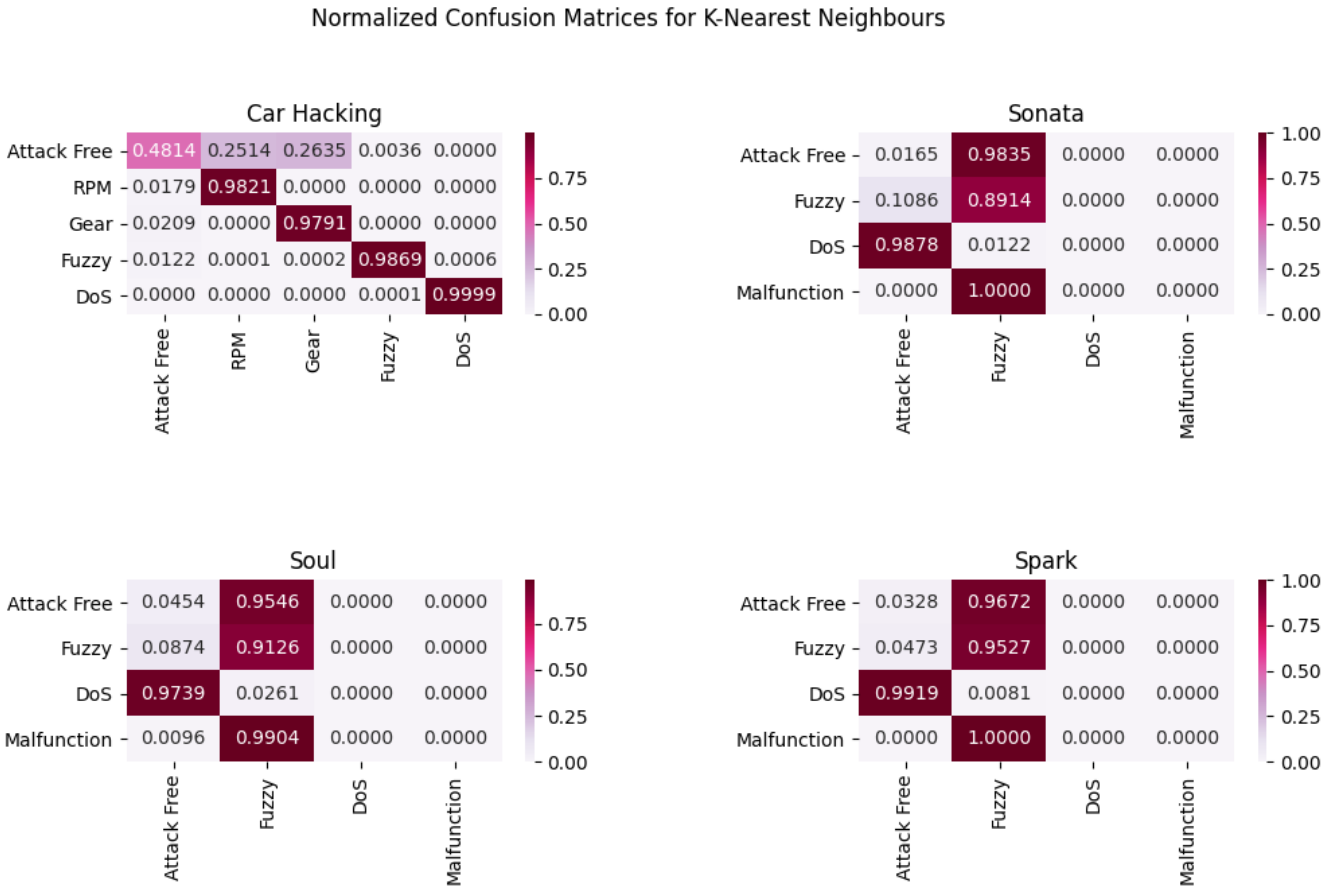


Fig 14. Normalized Confusion Matrices per dataset for the K-Nearest Neighbours classification method.

In the above Confusion Matrices, K-Nearest Neighbours has very good performance on all attack types for the Car Hacking set, only struggling with the benign data. Performance on the Sonata, Soul and Spark sets is very good for Fuzzy attacks only, while being nearly completely incorrect for Attack Free, Malfunction and DoS attacks.

5.3.2.4 Gradient Boosting

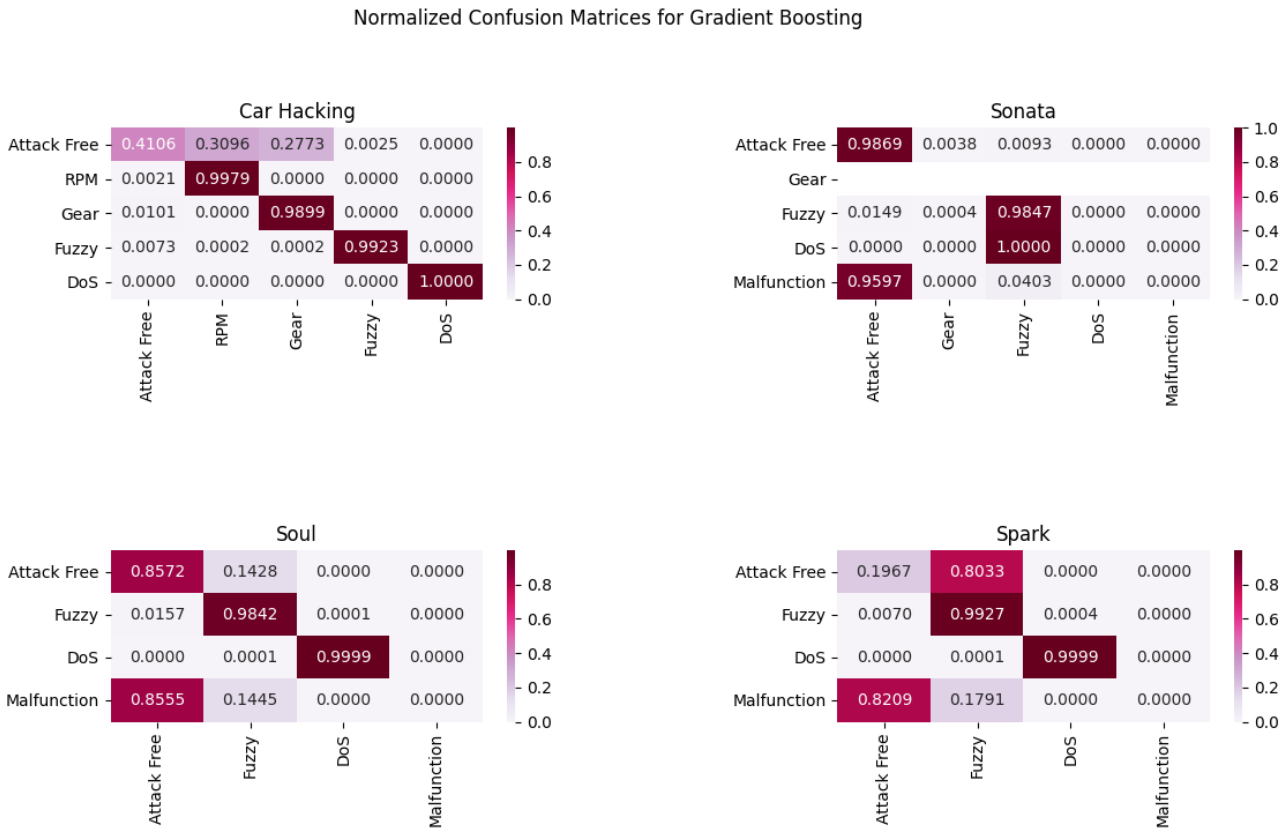


Fig 15. Normalized Confusion Matrices per dataset for the Gradient Boosting Classification method.

In the above Confusion Matrices, it can be seen that Gradient Boosting has very good performance on all attacks for the Car Hacking set, only struggling with the benign data. Performance on the Sonata, Soul and Spark sets is generally good, with near perfect scores occurring for each category save malfunction in 2 of the 3 cases. In every case, performance with Malfunction attacks is extremely poor.

5.3.2.5 Logistic Regression

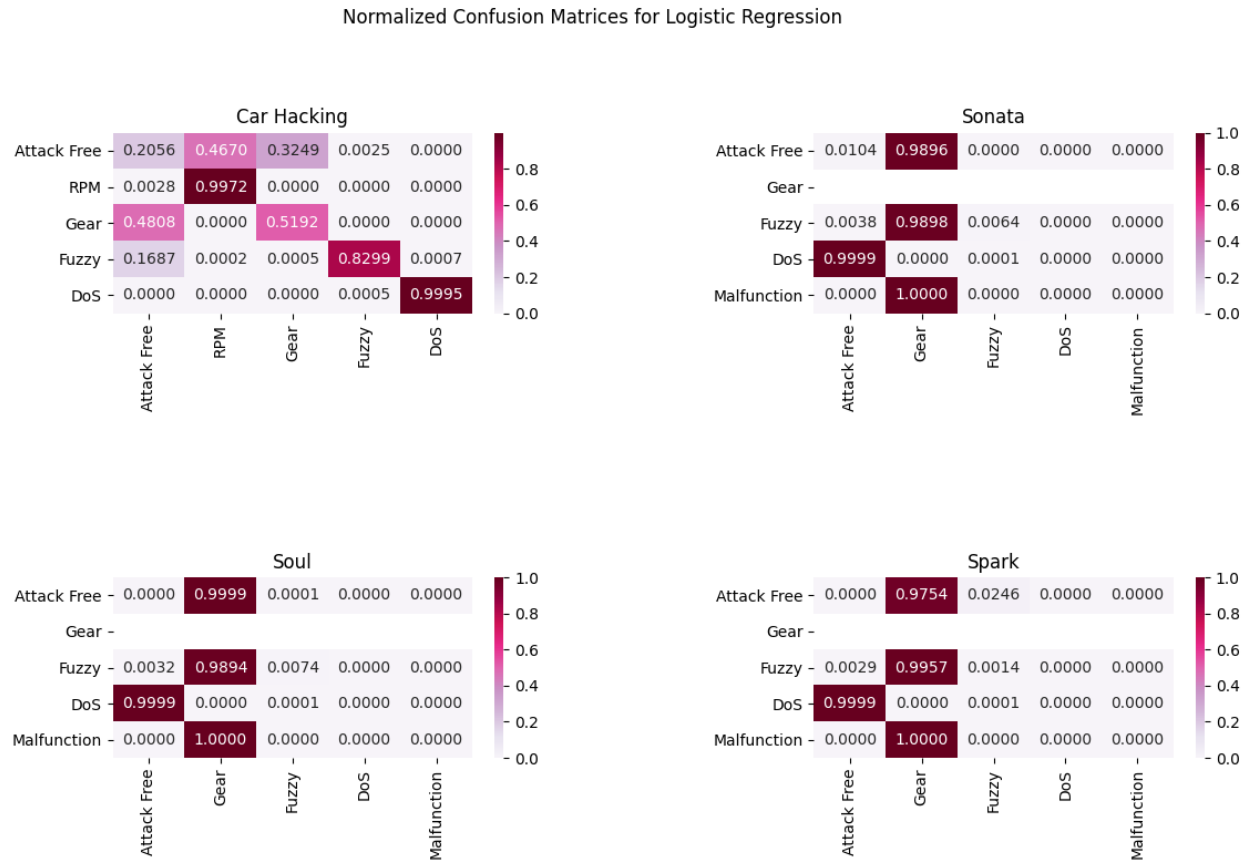


Fig 16. Normalized Confusion Matrices per dataset for the Logistic Regression classification method

In the above Confusion Matrices, Logistic Regression has very good performance on RPM and DoS attacks for the Car Hacking set, good for Fuzzy attacks, struggling with the benign data and Gear spoofing. Performance on the Sonata, Soul and Spark sets is almost universally incorrect, with everything targeted towards Gear spoofing (which is not present).

5.4 System Footprint

When considering how suitable any system is for the in-vehicle network, it is important to consider the potential limitation that is available on computing resources. As such, potential solutions should aim to have the lowest footprint possible, so that they are more easily translated to being implemented in the resource constrained environment. Below is a table that details the memory required to store each of the trained classifiers.

Table 6. Memory usage by classifier for second stage detections

Classifier	Memory Usage (MiB)
K-Nearest-Neighbours	319.660
AdaBoost	39.258
Isolation Forest	40.137
Linear SVC	35.305
LOF	48.996
OCSVM	35.633

As can be seen from the above table, in terms of resource usage, any of the techniques that are not K-Nearest Neighbours show very little difference in the amount of memory consumed. As such, this becomes a negligible factor when choosing between them and other factors are better suited for determining which is the better choice.

The same applies for the classification techniques as it does for the detection of attacks, that the least number of resources used is desirable. Below is a table detailing the memory usage of the different classification strategies.

Table 7. Memory usage by algorithm for the classification stage.

Classification Algorithm	Memory Usage (MiB)
Linear SVC	35.566
K-Nearest-Neighbours	319.163
Random Forest	1218.246
Gradient Boosting	41.008
Logistic Regression	34.848

As can be seen from the above table, Random Forest uses a very large amount of memory when compared even to K-Nearest Neighbours for the same training data, and K-Nearest Neighbours roughly 8-10 times more than all the other methods. While this is not a positive trait, it could be permissible if those methods show marked improvement over the others in the other areas.

5.5 Speed

The table below details the time taken by each possible combination of second stage and classification components, starting from a raw dataframe of 10000 CAN frames and ending after attack classification. The test dataframe is known to contain attack instances, so all stages of the system are needed.

Table 8. Time taken for each combination of second stage and classification step.

Second Stage Detector	Classifier	Time Taken (seconds)
Linear SVC	**Linear SVC**	**0.547**
Linear SVC	Random Forest	1.568
Linear SVC	**K-Nearest Neighbours**	**0.924**
Linear SVC	**Logistic Regression**	**0.544**
Linear SVC	**Gradient Boosting**	**0.570**
Isolation Forest	**Linear SVC**	**0.702**
Isolation Forest	Random Forest	1.713
Isolation Forest	K-Nearest Neighbours	1.384
Isolation Forest	**Logistic Regression**	**0.644**
Isolation Forest	**Gradient Boosting**	**0.678**
Local Outlier Factor	**Linear SVC**	**0.684**
Local Outlier Factor	Random Forest	1.912
Local Outlier Factor	K-Nearest Neighbours	1.864
Local Outlier Factor	**Logistic Regression**	**0.717**
Local Outlier Factor	**Gradient Boosting**	**0.778**
K-Nearest Neighbours	Linear SVC	1.849
K-Nearest Neighbours	Random Forest	2.800
K-Nearest Neighbours	K-Nearest Neighbours	3.168
K-Nearest Neighbours	Logistic Regression	1.822
K-Nearest Neighbours	Gradient Boosting	1.835
AdaBoost	**Linear SVC**	**0.610**
AdaBoost	Random Forest	1.644
AdaBoost	K-Nearest Neighbours	1.073
AdaBoost	**Logistic Regression**	**0.596**
AdaBoost	**Gradient Boosting**	**0.594**
One-Class SVM	**Linear SVC**	**0.540**
One-Class SVM	Random Forest	3.886
One-Class SVM	K-Nearest Neighbours	2.636
One-Class SVM	**Logistic Regression**	**0.546**
One-Class SVM	**Gradient Boosting**	**0.650**

From the table, it can be seen that most algorithms have a very low running time, but Random Forest and K-Nearest Neighbours are consistently slower than all other options. When used together, or with K-Nearest Neighbours twice, this starts to become slow in real-time terms, taking more than the time between the maximum time between message groups of this size to process. All other options would be viable choices and their suitability should be determined by other factors.

6. Evaluation of Results

6.1 Overview

This section will cover the interpretations and possible causes and implications of the results from the previous chapter, as well comparing the results obtained in this work to some of the work covered in section 2 to better understand the meaning of the work in context. This will also cover some of the reasons behind the choice of the parameters for the algorithms used.

6.2 Optimization

6.2.1 Second Stage

From the tuning that was carried out for each of the six algorithms, there are a few trends between them that emerged. Firstly, all of the unsupervised methods did not tend to see a performance improvement when scaled in the same way that the supervised methods did. This would suggest that the issue with their performance is not related to a lack of trees/neighbours/support vectors, but rather by another factor, which is possibly the dataset used for training. The chosen parameters for each of the algorithms were the best set of those that were tested, as seeming to vary them did not yield much difference in the results obtained on the data. The unsupervised algorithms are all trained from the attack free data, which would suggest that the provided data either does not represent the problem of attack detection, there is a lack of data for the chosen methods, or the methods chosen are not fit for use in this particular context. Of these, the last option is the most likely, as other works such as the GAN based system developed by Seo et al. used exactly the same dataset to train their system (which was able to achieve a detection rate of 98%), so it is unlikely to be the fault of the data.

The Linear SVC showed very small performance variation between 5-10 times bias towards attack instances, and then after 10 times bias, showed a rapid drop in performance on the test set as it began to overfit. This is why the bias was fixed at 7 times, as this was the rough middle ground between 5 and 10, with the primal problem being preferred for computational efficiency. When using the AdaBoost algorithm and K-Nearest Neighbours in the distance-based option, the performance increased as more iterations or neighbours were added, though this exhibited diminishing returns once certain thresholds were reached. It is for this reason that the chosen parameters for these algorithms are fixed where they are, as past these points is where the returns decrease and an undesirable characteristic increases. In the case of K-Nearest Neighbours, this is execution time and in the case of AdaBoost, the memory usage.

6.2.2 Classification

For the classification algorithms, the trend was the same across all of those who had a parameter to vary. As the number of estimators or iterations increased, the performance of the algorithm also did. However, much like with the second stage components, this comes at a trade off with another factor. For the Random Forest and K-Nearest Neighbours algorithms, this is memory usage, as more memorized training data or trees in the forest were needed to improve the performance. K-Nearest Neighbours also slows down the more neighbours that are used for each query, which would reduce the effectiveness of the classifier, so a balance between the two was picked with the number of 10. For Gradient Boosting, Logistic Regression and the Linear SVC, the way to improve their performance would be to use more

training data or allow for more iterations. In each case, this will increase the size of the resulting classifier, which may not prove to be worth the trade-offs in performance, as the gains achieved tended towards diminishing returns after a certain point. Logistic Regression did not see much variation in performance between the changed parameters. From the API page, the SAGA solver offered fast convergence for scaled data, which is why it was chosen. The values for max_iter were determined to be necessary to avoid a warning for not using enough iterations. The l2 penalty was chosen as it showed better performance on tests than the l1 penalty.

6.3 Identification Performance

6.3.1 Second Stage

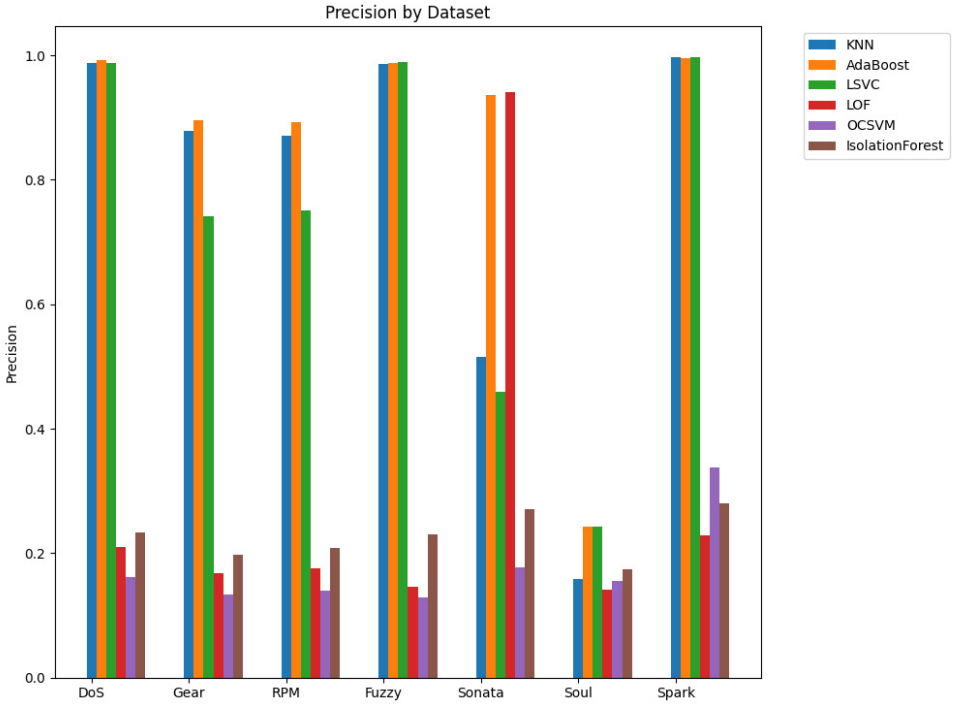


Fig 17. Bar Graph showing the Precision of each algorithm by dataset

The above bar graph is constructed from the Precision data in Table 5. From it, the following conclusions can be drawn. Near universally, the unsupervised algorithms show very low Precision when compared to their supervised counterparts, with the sole exception being in the performance of LOF on the Sonata dataset (the reason for this is not known). Generally, the Isolation Forest performs the best of the unsupervised techniques, but still has a false positive rate of approximately 80%. This makes it and the other worse performing unsupervised algorithms unfit for the task they have been designed to perform. As mentioned previously, this could be because of the datasets used to train them, but it is more likely to be the fault of the algorithm/feature combination used, as others can achieve good results with a neural network trained using the same data.

Of the supervised algorithms, the AdaBoost algorithm consistently achieves the best, or near to it across all of the involved datasets, with K-Nearest Neighbours often very similar, though slightly worse performing. The LSVC performs well on the DoS and Fuzzy sets (and by extension the Soul and Spark sets) but has trouble with the RPM and Gear spoofing detection,

seeing a sharp increase in false positives (though other algorithms see the same, it is not as severe). This would suggest that these instances are hard to separate using a linear model and require a more complex detection or more features to be able to improve performance.

All algorithms performed poorly on the Soul dataset, with some variation. This is likely because of one of two things. Either the training data used does not properly model the problem, or it is the rule-based component (the timing component especially) causing such a difference in the false positive rate, with the variance being down to the algorithms themselves. Of these two options, the latter is more likely as the work of Zhang et al. used the same datasets with their combination rule based/DNN system and were able to achieve much, much lower false positive rates, being less than 0.01%.

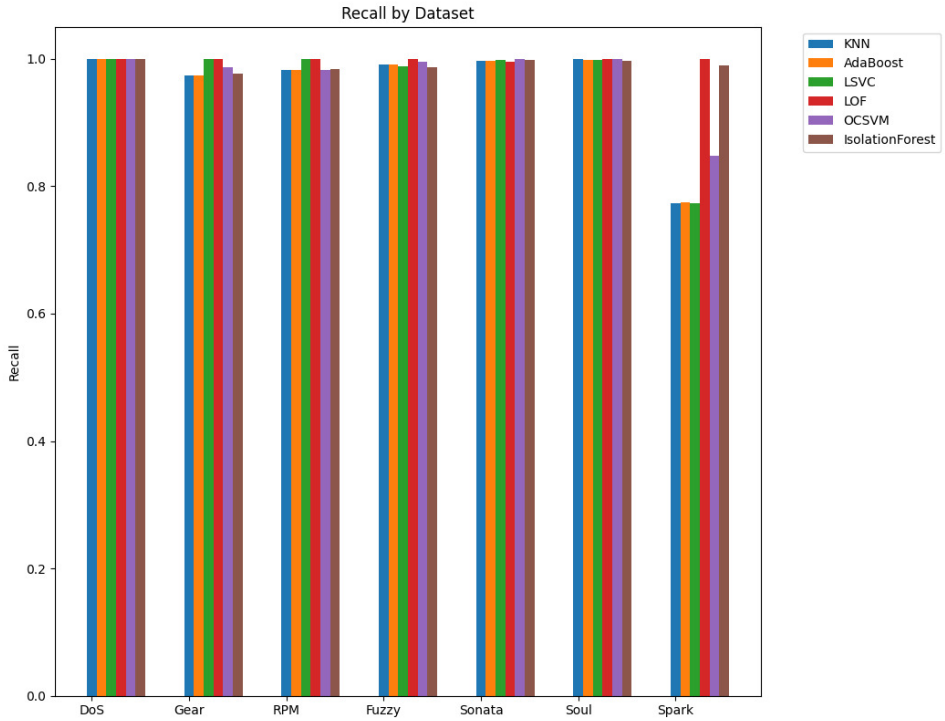


Fig 18. Bar Graph showing the Recall of each algorithm by dataset

The above bar graph is constructed from the Recall data in Table 5. From it, the following conclusions can be drawn. All algorithms achieve perfect Recall on the DoS dataset, which means that the algorithm used has no effect on it. This can be confirmed to be the case if the ID rule is followed through logically. The Dos attacks all have ID 0x0000, which is not a recognized ID, so is immediately filtered out. For all other datasets, very high Recall scores (refer to Table 5 for exact values) are achieved by all algorithms, often with the unsupervised algorithms slightly edging out the supervised ones, though this comes at the cost of a lot of Precision. The detection performance is similar to many of the methods employed in the existing work in the field, with only some methods slightly beating out the ones used here by 1-2%. A notable point is that for the Spark dataset, the unsupervised algorithms caught some or all of the Malfunction attack, whereas the supervised algorithms did not (which is why they all show much lower Recall). This is likely because they were not similar enough to the known attack classes to be marked as attacks, and so passed undetected.

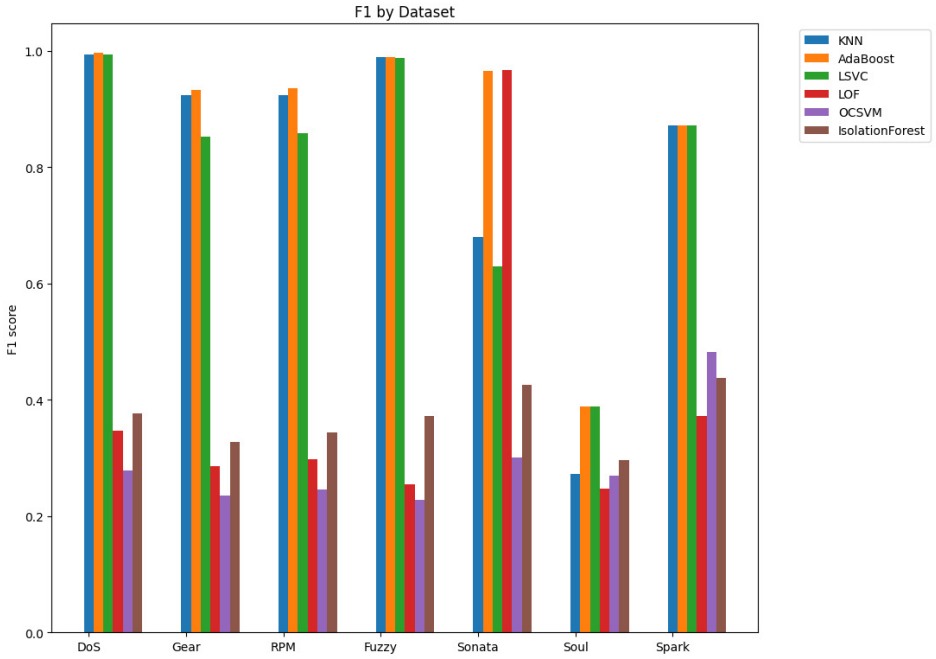


Fig 19. Bar Graph showing the F1 score of each algorithm by dataset

The above bar graph is constructed from the Recall data in Table 5. From it, the following conclusions can be drawn. With F1 scores consistently in the 0.2-0.4 range, all of the unsupervised algorithms are not fit for purpose. While they are able to achieve very high rates of detection, this comes at an unacceptable false positive rate with the implementation as is. It must either be refined with additional features or other improvements, or different unsupervised learning techniques used, such as that of the GAN by Seo et al. For the supervised learning methods, K-Nearest Neighbours and AdaBoost are often very similar, with AdaBoost beating out K-Nearest Neighbours by small amounts in most cases. As the detections from the rule-based components are constant, the difference in performance must come from the algorithm used, which allows the conclusion that AdaBoost is a better algorithm to use. The LSVC is able to achieve similar detection performance but is held back by the higher false positive rates, making it less suitable than either K-Nearest Neighbours or AdaBoost. It is not known why LOF achieves very high performance on the Sonata set when it is normally poor otherwise.

6.3.2 Classification

Referring to the Confusion Matrices in the Results section (5.4.2.x, Figs 12-16), it can be seen that the classifiers often struggle to determine the false positives generated by the previous steps, with more often than not them getting misclassified as a type of attack. On the Car Hacking dataset, discounting the misidentification of normal instances (most often as a type of spoofing, likely because they share the same CAN ID and Relative Entropy values), all of the algorithms used achieve a high performances in classifying attacks correctly. DoS attacks are almost universally identified correctly, with several classifiers achieving perfect scores and the maximum error being 0.05% using Logistic Regression. This is likely because they all share a single defining, unique characteristic that is not present anywhere else, a CAN ID of 0x0000, which is easy to differentiate. Fuzzy attacks are generally identified well, though the Linear SVC and Logistic Regression methods are noticeably worse, with an

accuracy of only 82.99%, rather than the 98.69-99.7% range achieved by the other algorithms, with the difference being in them being misclassified as benign. For RPM spoofing, all algorithms perform well, ranging from 97.7% to 99.7% correct identifications. For Gear spoofing, the Linear SVC and Logistic Regression methods are very poor, barely reaching over 50% accuracy, but the remaining algorithms are much better, scoring between 97.37% and 98.99%. For both spoofing methods, it is believed that the reason why the simpler models perform much more poorly is that they simply do not have the required depth and resolution to properly differentiate the spoofing instances from the normal instances of the same CAN ID that have made it to the classifier.

On the Sonata dataset, the only attack that is consistently classified correctly is the Fuzzy attack, with the Linear SVC seeing the worst performance in this case, though other algorithms had very good scores. DoS attacks were consistently misclassified as either normal or Fuzzy instances, depending on the classifier used, with only Random Forest actually performing as desired. It is not known why this behaviour occurs, as the DoS attacks from the Car Hacking set (used in training) should be similar enough to still achieve a good classification result, though this may not be the case from the observed results. Malfunctions were universally classified incorrectly, but this was somewhat expected behaviour, as they were deliberately left out of training to see how the system would react to unknown attacks, it is not surprising that the classifier tries to fit them into one of the known classes instead.

On the Soul dataset, Fuzzy attacks are again classified very well, with only the Linear SVC seeing poor performance at around 50%, and Logistic Regression completely misclassifying them as Gear spoofing. DoS attacks were identified correctly by Gradient Boosting, Random Forest, and Linear SVC, with accuracy levels similar to that of the Car Hacking dataset. This is noteworthy as the same classifiers (as they are not retrained) failed to identify them correctly with the Sonata set, so there must be a difference in one of the features that causes the change. As the CAN ID and Time Intervals should be very similar, it is either the way they cluster together in the data (affecting the count in last 1000 messages), or the overall change in entropy that is causing the difference. Logistic Regression and K-Nearest Neighbours misclassified them entirely as normal behaviour. Malfunctions were again classified incorrectly, with most classifiers opting to label them as Fuzzy attacks, though some were labelled as normal instances or Gear spoofing. As mentioned, when discussing the Sonata set, this is expected behaviour from the system.

On the Spark dataset, Fuzzy attacks are also classified well by the same classifiers that have performed well on the other sets, with the Linear SVC and Logistic Regression being the ones that are misclassifying them. DoS attacks were classified correctly by Gradient Boosting and Random Forest, with other classifiers labelling them as normal data or Malfunction instances. The actual Malfunction instances were labelled as normal, Fuzzy or Gear spoofing, which is to be expected as has been previously mentioned.

Overall, it can be said that while training a more general- purpose classifier using a variety of attacks is possible, this can lead to unintended behaviour when applied to a different vehicles' environment. Some classifiers are more sensitive to this than others, with the Linear SVC, K-Nearest Neighbours and Logistic Regression more sensitive to the changes than the Random Forest or Gradient Boosting methods. For K-Nearest Neighbours, this is expected, as the data that it is seeing does not match what the classifier has trained on and memorized, which

makes the algorithm a poor choice for portability between systems. For the Linear SVC and Logistic Regression, it is probably because the learned function is fitted well to the particular training data, but because of the lack of standards between CAN networks, it is not greatly portable. Random Forest and Gradient Boosting on the other hand, do not see the same issues, as they were able to correctly classify all or almost all of the attacks in the other datasets without being retrained. The difference between the two is that Gradient Boosting was more able to recognize the normal instances (i.e. false positives from the second stage), whereas the Random Forest labelled these all as Fuzzy attacks. This is likely because the CAN IDs do not match up to the ones that form the decision trees of the forest, so get labelled as Fuzzy attacks. It should be noted however, that this performance will only hold true for datasets that do not also have other types of spoofing present. As the ability to recognize spoofing attacks is dependent on the CAN ID used for the component and these IDs are all non-standard, it becomes impossible to develop a portable classifier that includes spoofing-type attacks as it relies on the specifics of the particular network it would be used on. Therefore, it will be more practical to retrain the classifiers across different networks, rather than use a general purpose one, unless there are some form of standards adopted for the use of CAN ID values.

6.4 System Footprint

As can be seen from Table 6, most of the chosen classifiers have little difference when it comes to memory usage, save for K-Nearest Neighbours, which must memorize its training data. However, if how usage may change with different parameters is considered, some scale better than others. The Linear SVC needs only store a mathematical function, regardless how of many points it is trained with, the memory usage will not increase much, if at all, unless more features are added (as this adds coefficients in the primal problem). Isolation Forest and LOF both scale logarithmically with training size, as they both rely on tree structures to store their data internally, so can handle large volumes of train data if necessary. AdaBoost will scale logarithmically with more complex training data (as the base classifier is a decision tree in this case), or linearly if the number of estimators is increased, as this adds more trees. OCSVMs scale well in terms of space, as they are learning a decision boundary, though their size is determined by the number of required support vectors. As such, any of the methods that are not K-Nearest Neighbours have excellent or good space usage and scaling so other factors should be more important when choosing the algorithm to use.

Table 7 shows a similar trend for the classifier choices. While Random Forest the most effective classification method, it comes with the drawback of very large amounts of memory usage, even with a relatively small forest. Considering that in real network, there may be tens or maybe even hundreds of unique ECUs that all need to have spoofing attempts detected, it is very possible that a Random Forest for this situation would be prohibitively large, as it is likely that the amount and/or depth of the trees would need to increase to describe the data correctly. The usage for the K-Nearest Neighbours algorithm is also relatively high and would suffer from the same issue as the Random Forest, if not even worse. The K-Nearest Neighbours system would need to memorize a set of points for each of the classes to be able to use, so if the number of classes gets larger, so does the required amount of memory, which limits its usefulness in large networks.

The Gradient Boosting and Logistic Regression methods both show minimal memory usage, though this is for different reasons. The Gradient Boosting system is low because there were

few iterations needed to show a very high performance and the number of overall attack classes was low. This reduced the number of weak classifiers the system used. In a more realistic network with tens or hundreds of classes, the usage would increase (but rather slowly) as more classifiers must be fitted per boosting iteration. Logistic Regression scales the best of the classifiers used, as it needs only learn a single mathematical function, which will always have a very low memory footprint. Unfortunately, the performance is the worst of the tested classifiers, with the savings in memory not worth the trade-off in classification ability.

6.5 Speed

When considering the speed required of the system, the context is important. For the implementations here, the target specification was CAN 2.0A with a bus speed of 1Mbit/s (maximum). This means that potentially, there can be 21276 frames per second (the calculation of why is below). This calculation makes two assumptions, namely the bus is under maximum utilization and that there are no bits added by bit stuffing (though this would never be the case in a real system, meaning less messages per second are needed). However, most CAN frames (at least in the datasets used here) were of maximum size, so 9009 frames per second was a much more realistic estimate, though 10000 was used to allow for some smaller frames.

$$1Mbit = 1{,}000{,}000\ bits$$

$$Minimal\ Size\ Frame = 1 + 11 + 1 + 1 + 1 + 4 + 0 + 15 + 3 + 7 + 3 = 47bits$$

$$Maximum\ Size\ Frame = 1 + 11 + 1 + 1 + 1 + 4 + 64 + 15 + 3 + 7 + 3 = 111bits$$

$$Maximum\ Minimal\ Size\ Frames\ Per\ Second = \frac{1000000}{47} = 21276\ frames$$

$$Maximum\ Maximum\ Size\ Fames\ Per\ Second = \frac{1000000}{111} = 9009\ frames$$

Referring to Table 8 in section 5.5, it can be seen that any approach utilizing K-Nearest Neighbours (with the exception of the Linear SVC/K-Nearest Neighbours combination) or Random Forest takes greater than one second to perform its task and as such, is too slow for use in the proposed scenario. While these algorithms could be increased in speed, the trade-off is in performance, as this would require reducing the number of queried neighbours or the number of trees in the forest, respectively. For the combinations that take less than one second, there are the options of using the Linear SVC, Isolation Forest, Local Outlier Factor or AdaBoost for the detection stage and either Logistic Regression or Gradient Boosting for the classifier.

6.6 Summary

To sum up, while all of the tested detection algorithms were able to achieve very high rates of Recall (i.e. detection rate), the One Class SVM, Isolation Forest and Local Outlier Factor did this as the cost of a very substantial amount of Precision, which limits their overall usefulness due to the amount of false positives generated. The K-Nearest Neighbours, Linear SVC and AdaBoost algorithms did not see the same issue with Precision, and therefore were able to achieve very high F1 scores. Of these, the AdaBoost algorithm was the algorithm that consistently performed the best, only not being so on one of the seven tested datasets.

In terms of classification algorithms, Logistic Regression and Linear SVC performed consistently poorer than K-Nearest Neighbours, Random Forest, or Gradient Boosting. All tested classifiers performed more poorly on the datasets where they had not been trained on any data from it, due to the difference in the IDs present. The attack that was not part of the training was consistently mislabelled, though this was fully expected to be the case. Overall, the Random Forest and Gradient Boosting were the two algorithms that showed the best performance. It would also be better to train the classifiers with samples from the networks that they would defend, as the differences between IDs caused some issues that would hamper attempts to use a general-purpose classifier, especially in the context of spoofing type attacks.

With regards to the speed and resource usage of the algorithms at both stages, K-Nearest Neighbours and Random Forest were the two algorithms that were not fast enough for using in a real-time context and also showed much higher memory usage than the other algorithms, leading them to be less desirable or fully impermissible choices. Considering all of the above factors, the best performing algorithms were the AdaBoost for detection and the Gradient Boosting for classification.

7. Conclusion

In conclusion, this work shows that it is possible to create a system for increasing the awareness of and detecting attacks in the in-vehicle network, while also classifying those attacks by type. The data is first pre-processed, with some of the bus data removed and additional features added, before it is passed through a two-stage detection system, with the first stage being rule-based and the second a machine learning algorithm. The best approach tested was the AdaBoost meta-strategy, using a decision tree as the base learner, able to achieve an average detection rate of 95.95% across 5 attack types, including one that it was not trained on. For the frames that are detected as attacks, the classification component is able to sort these by attack type or false positive with a high degree of accuracy, the best performing algorithm was the Random Forest, with an average classification for recognized attacks of 99.40% across all tested datasets. However, this comes at the cost of a run-time and memory consumption far higher than other algorithms. A Gradient Boosting approach (using a regression tree as the fitted model) was able to achieve 89.42% across all datasets, with a runtime suitable for real time use and using 30 times less memory than the Random Forest. These statistics include the classifiers being applied to datasets they were not trained on (with the difference coming from the Gradient Boosting misclassifying the DoS attacks in one set, without this the accuracy would be 99.5%). However, it is not recommended to take this approach when using the system to classify spoofing, as the classifier must learn the IDs specific to that network, the classifier should be trained using the system it would protect. The combination of AdaBoost detection and Gradient Boosting classification together only take 0.594 seconds to process a frame of 10000 messages, a per message time of 59.4 microseconds, which is sufficient for use in a real-time context on a CAN 2.0A bus with a bus speed of 1Mbit per second.

8. Possible Future Work

From this work, there are multiple possible extensions that could be made. Firstly, there is the option to add more rules to the rule-based component, or to implement improvements to the existing rules. A possible example of this could be a refinement to the timing rule that is able to adjust the recorded interval if there is a suspected attack between two frames that are sent regularly. There is the option of using different algorithms for the two latter stages, such as the neural networks used by other authors in their own work to achieve very high accuracy at the detection of attacks, though (to the author's knowledge) there is not any other existing system for also classifying the attacks.

It is also possible to consider applying the system as is to an attack model that is more representative of real-world vehicle threat environment, where there are more variations on the spoofing and DoS type attacks, as well as classes of attack that the system may not have seen before. This would also help to determine the feasibility of capturing traffic and performing the attacks against the network required to train the algorithms that learn in a supervised manner. In conjunction with this, it would be beneficial to attempt to implement the system in terms of ECU and/or microcontroller hardware.

Another possible direction for future work is extending the system to be able to work with the different kinds of in vehicle networks. This not only includes different variants of the CAN bus itself, but also a version for a LIN bus or FlexRay, in order to protect every aspect of the network from intrusion. While LIN systems are not safety critical, attackers can still cause a nuisance or prove an irritant by compromising these systems

It could be possible to extend the classification element of this work into a prevention system, where the system detects and then takes active countermeasures against the detected attacks, such as by disabling the communication of a believed compromised ECU so it can no longer inject messages into the network. As the attacks are classified by type, there is the potential to take different measures against different kinds of attacks if that is the desired policy.

Finally, it could also be possible to use the system as a way of collecting big data to provide insights about the kinds of threat the system faces, and which parts of the networks should receive closer examination, patches, or additional hardening measures. This would require the system to be deployed across a number of vehicles in a real context and have the ability to record and communicate information to a central server, so is most outside of the scope of the work carried out but is still technically feasible.

9. Reflection on Learning

Over the course of conducting this project, I have learned a number of things, ranging from knowledge of the specific domain to more general knowledge about the field of neural networks and machine learning in general. Prior to starting the research and the practical components of it, the only prior experience I had with the area came from a YouTube channel, CodeBullet, who uses neural networks to play games or to learn a specific task (though he often uses reinforcement learning methods, such as Deep Q learning) and from my coursework for Large Scale Databases, in which I used the SpaCy and Flair modules for Python as part of the Spatial component of the coursework assignment. I now know that the model I created (these modules rely on PyTorch to train bidirectional LSTM networks) as part of this assignment were almost certainly overfitted to the training data that I provided to it (as I performed a naïve method of oversampling, as only 12 sentences were provided). Even though I did not end up using a neural network as part of the final implementation, they were still explored as an option and as part of the background reading undertaken.

Before I could conduct the research portion of this work, I first had to learn the underlying theory behind the in-vehicle network, as I had not even considered how this operated until taking on this project. Moreover, though not relevant here, I learned that there are multiple alternatives to the use of the CAN bus in more modern vehicles (LIN, FlexRay and Automotive Ethernet) that offer advantages when compared to using the CAN bus alone. As part of the knowledge required to understand how the existing work in the field operated, in addition to understanding how the CAN bus operated, I also has to have at least a high-level understanding of the various techniques that were being used to provide IDS functionality to the bus. Therefore, I now know conceptually how multiple kinds of neural networks operate, with some of the associated advantages and disadvantages of each, as well of some of how to implement them (though neither implementation was successful). Furthermore, I can say how these are applied to the specific domain of the in-vehicle network. As well as the neural networks, I have also learned about the other forms of machine learning algorithms, that I actually then came to use in my own work. This includes the knowledge of how they work conceptually, how to tune and optimize the various parameters that are unique to each algorithm, with some of the potential consequences of fixing these too high or low and when they might be better suited to other use cases.

As well as having learned about the algorithms themselves, I have also learned about the various ways that the performance of them can be quantified or illustrated. From the work in Large Scale Databases, I was familiar with the concepts of Precision, Recall and F1 scores, but confusion matrices and other measures were all new to me. I have also learned the importance of either using cross-validation or a test/validation/test split in the dataset when it comes to assessing how well your algorithms are performing, as without this you can see very good results because the algorithm is able to memorize the training data.

The time requirement of the in-vehicle network made me focus on the performance of my code, which is not something that I have ever had to do to such an extent before. Normally, a solution in finite time would be good enough for my purposes, as long as it delivers on the desired result (and does not cause high amounts of framerate drops in the case of code with graphics components). However, this project required a fair amount of optimization in certain

areas to achieve a runtime that would be usable in the intended context, which helped me develop two important skills. The first of these is profiling my code using an appropriate module and the second is that certain ways of doing something are just inherently better than others (this mainly refers to using numpy and vectorized functions rather than a pandas apply where possible).

This has also been the biggest experience that I have had to date with producing a large formal piece of work. It has been beneficial in the realms of time management, work-life balance and the ability to write in the particular style that is required for something like this. Furthermore, it has reinforced the importance of making sure that I step back and consider the potential reasons and justifications (though I feel that some of the ones used here could definitely be improved) for using one method over another, rather than just iteratively trying things out until I find the method that works the best. Finally, it has helped me be able to impose a higher level of self-discipline when undertaking something as large as this with less support than I have been used to in the past such as at A-level, which was the last time I did a large piece of programming work that also needed a formal style report to go alongside it.

A. References

[1] Miller and Valasek, Remote Exploitation of an Unaltered Passenger Vehicle, available at http://ericberthomier.fr/IMG/pdf/remote_car_hacking.pdf

[2] Bozdal et al, A Survey on CAN Bus Protocol: Attacks, Challenges and Potential Solutions. 2018 International Conference on Computing, Electronics and Communications Engineering. Available at https://core.ac.uk/download/pdf/287585022.pdf

[3] Upstream Automotive, Global Automotive Cybersecurity Report 2020, available at https://info.upstream.auto/hubfs/Security_Report/Security_Report_2020/Upstream%20Security-Global_Automotive_Cybersecurity_Report_2020.pdf?_hsmi=80788392&_hsenc=p2ANqtz-_rV_dwa7fYay2CbYW-iJ1nrKXzqn9j7m035B0R29FxsMSACLTqoQ5sAkMm_U_dyQ4dFB5RsFPqu3Fq3sX1vSClkCJtTw

[4] Siddiqui et al (2017). Secure communication over CAN Bus. 1264-1267. 10.1109/MWSCAS.2017.8053160. Available at https://www.researchgate.net/publication/320607413_Secure_communication_over_CANBus/

[5] M. Jukul and J. Čupera, Using of tiny encryption algorithm in CAN-Bus communication. Available at https://www.agriculturejournals.cz/publicFiles/12_2015-RAE.pdf

[6] A.I Radu and F.D Garcia, LeiA: A Lightweight Authentication Protocol for CAN, available at https://www.cs.bham.ac.uk/~garciaf/publications/leia.pdf

[7] M. Müter and N. Asaj, "Entropy-based anomaly detection for in-vehicle networks," 2011 IEEE Intelligent Vehicles Symposium (IV), 2011, pp. 1110-1115, doi: 10.1109/IVS.2011.5940552.

[8] J. Li, Deep Learning on CAN bus, available at https://www.youtube.com/watch?v=1QSo5sOfXtI&feature=youtu.be

[9] K.T Cho and K.G Shin, Fingerprinting Electronic Control Units for Vehicle Intrusion Detection. Available at https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_cho.pdf

[10] Arliou Sentinel-CAN. System overview at https://ariloutech.com/solutions/sentinel-can-intrusion-detection-prevention/

[11] M. Marchetti and D. Stabili, Anomaly detection of CAN bus messages through analysis of ID sequences. Available at https://iris.unimore.it/retrieve/handle/11380/1149168.1/173601/FINAL_SUBMISSION.pdf

[12] H.M. Song et al. Intrusion Detection System Based on the Analysis of Time Intervals of CAN Messages for In-Vehicle Network. Available at https://awesong-kor.github.io/files/Intrusion%20Detection%20System%20Based%20on%20the%20Analysis%20of%20Time%20Intervals%20of%20CAN%20Messages%20for%20In-Vehicle%20Network.pdf

[13] E. Seo et al. GIDS: GAN Based Intrusion Detection System for In-Vehicle Network. Available at https://arxiv.org/pdf/1907.07377.pdf

[14] H.M. Song et al. In-vehicle network intrusion detection using deep convolutional neural network, Vehicular Communications, Volume 21,2020,100198, ISSN 2214-2096,

[15] G. Loukas et al. Cloud-based cyber-physical intrusion detection for vehicles using Deep Learning. Available at https://gala.gre.ac.uk/id/eprint/18213/7/18213%20LOUKAS_Cloud-Based_Cyber-Physical_Intrusion_Detection_2017.pdf

[16] H. Qin et al. Application of Controller Area Network (CAN) bus anomaly detection based on time series prediction, Vehicular Communications, Volume 27, 2021, 100291, ISSN 2214-2096,

[17] L. Zhang et al. A Two-Stage Deep Learning Approach for CAN Intrusion Detection. Available at https://events.esd.org/wp-content/uploads/2018/08/A-Two-Stage-Deep-Learning-Approach-for-Can-Intrusion-Detection.pdf

[18] M.J Kang and J.W Kang (2016) Intrusion Detection System Using Deep Neural Network for In-Vehicle Network Security. PLOS ONE 11(6): e0155781. https://doi.org/10.1371/journal.pone.0155781

[19] C. Young et al. Automotive Intrusion Detection Based on Constant CAN Message Frequencies Across Vehicle Driving Modes. Available at https://www.ece.iastate.edu/~zambreno/pdf/YouOlu19A.pdf

[20] M. Markovitz and A. Wool, Field Classification, Modeling and Anomaly Detection in Unknown CAN Bus Networks. Available at https://www.eng.tau.ac.il/~yash/escar-2015-10-13.pdf

[21] OneClassSVM, Available at https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html

[22] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

[23] C.C. Chang and C.J. Lin, LIBSVM : a library for support vector machines. ACM Transactions on Intelligent Systems and Technology, 2:27:1--27:27, 2011.

[24] LinearSVC, Available at https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[25] Fan, R.-E. et al., 2008. LIBLINEAR: A library for large linear classification. Journal of machine learning research, 9(Aug), pp.1871–1874.

[26] K. Crammer and Y. Singer, On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. Available at https://jmlr.csail.mit.edu/papers/volume2/crammer01a/crammer01a.pdf

[27] IsolationForest, Available at https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[28] KNeighboursClassifier, Available at https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[29] LocalOutlierFactor, Available at https://scikit-learn.org/0.20/modules/generated/sklearn.neighbors.LocalOutlierFactor.html

[30] AdaBoostClassifier, Available at https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

[31] RandomForestClassifier, Available at https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[32] LogisticRegression, Available at https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[33] GradientBoostingClassifier, Available at https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

[34] The pandas development team, pandas-dev/pandas: Pandas, located at https://doi.org/10.5281/zenodo.3509134

[35] W. McKinney, Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, Volume 445, 2010, pp. 56-61.

[36] Car Hacking Dataset, HCRL, Available at https://ocslab.hksecurity.net/Datasets/CAN-intrusion-dataset

[37] Survival Analysis Dataset for Automobile IDS, HCRL, Available at https://ocslab.hksecurity.net/Datasets/survival-ids